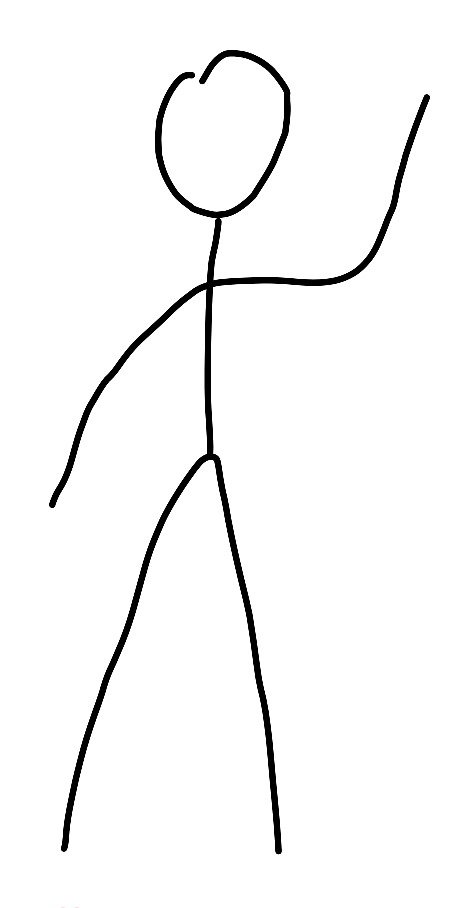

Архитектура надёжной In-Memory-СУБД
на примере Tarantool
Владимир Перепелица
HL
HighLoad++
2022
Яндекс


Архитектура надёжной In-Memory-СУБД
на примере Tarantool
Mons Anderson
HL
HighLoad++
2022
Яндекс
T

Кто я?

Архитектор Облака Mail.ru, VK Cloud (S3)

Архитектор и продакт-менеджер Tarantool

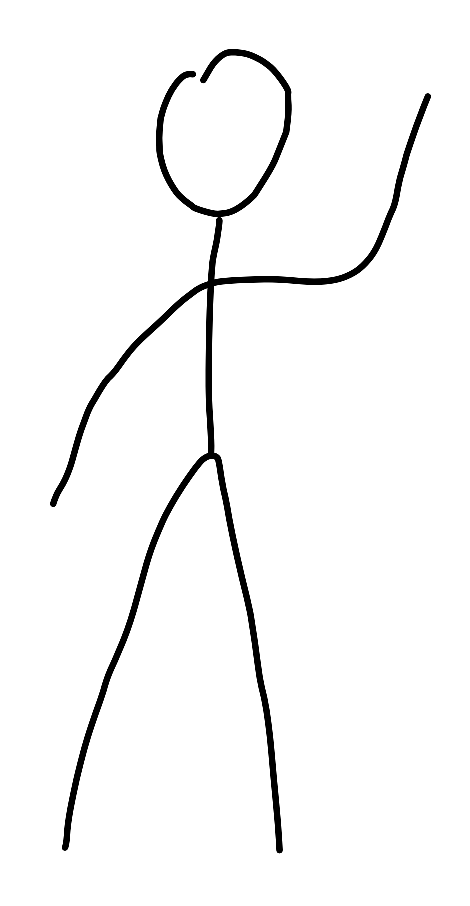


Кто я?

Архитектор Облака Mail.ru, VK Cloud (S3)

Архитектор и продакт-менеджер Tarantool

Евангелист Tarantool



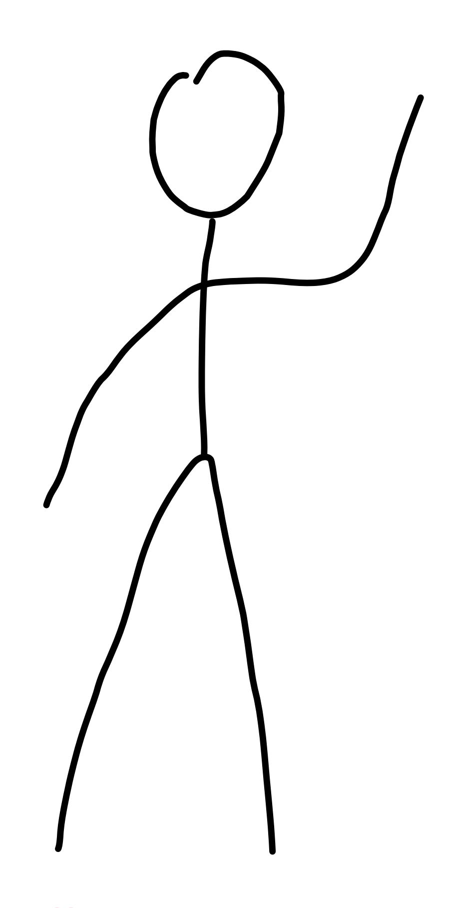
Кто я?

Архитектор Облака Mail.ru, VK Cloud (S3)

Архитектор и продакт-менеджер Tarantool

Евангелист Tarantool

Использую Tarantool более 10 лет



Кто я?

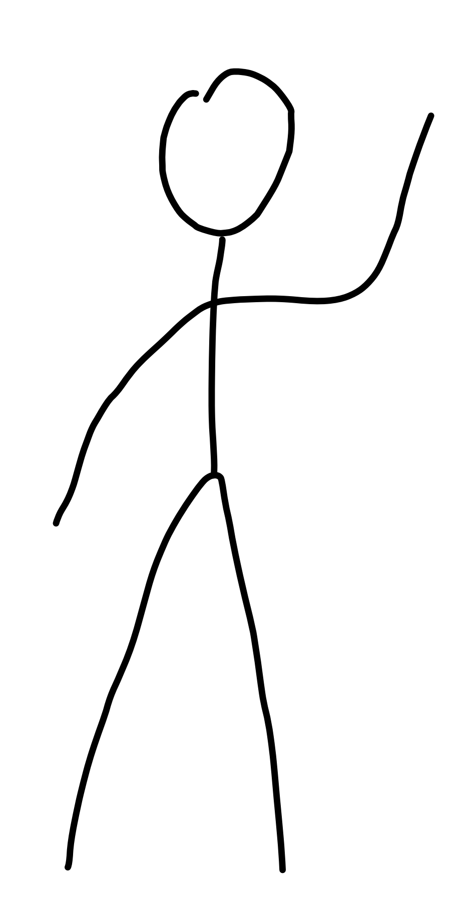
Архитектор Облака Mail.ru, VK Cloud (S3)

Архитектор и продакт-менеджер Tarantool

Евангелист Tarantool

Использую Tarantool более 10 лет

- Как основное хранилище
- Без «запасного» варианта рядом



О чём мы поговорим?

01 Почему In-Memory?

02 От кеша к БД

03 Персистентность

04 Репликация

05 MVCC

О чём мы поговорим?

01 Почему In-Memory?

02 От кеша к БД

03 Персистентность

04 Репликация

05 MVCC

Ссылка на слайды будет в конце

01

Зачем нужны In-Memory СУБД?

И почему не взять «старый добрый» SQL с кешом на тер?

Откуда взялись In-Memory БД?

- Изначально БД — дисковые
- И медленные
- Память — это кеш

Откуда взялись In-Memory БД?

- Изначально БД — дисковые
- И медленные
- Память — это кеш
- В начале 00-х производительности стало не хватать
- Появились первые решения для хранения в памяти





TimesTen: 1998 (!)

In-Memory RDBMS

Разработана в HP

Куплена Oracle в 2006

Проприетарная :(

137.	↓ 132.	↓ 122.	TimesTen	Relational	2.02	-0.07	-0.03





Memcached: 2003

Распределённая система кеширования в памяти

Для Livejournal

Простой API

Шардинг на клиенте

Автовытеснение

Кеш, а не надёжное хранение





Кто ещё?

TimesTen 1998

Memcached 2003

Hazelcast 2008

Tarantool 2008

Redis 2009

Aerospike 2012

Apache Ignite 2015





Кто ещё?

TimesTen 1998 (C)

Memcached 2003 (Perl/C)

Hazelcast 2008 (Java)

Tarantool 2008 (C)

Redis 2009 (C)

Aerospike 2012 (C)

Apache Ignite 2015 (Java)





Почему не <любимый-SQL> с кешом?

- Обычные реляционные БД ориентированы на диск
- Работа с диском не экономит размер
- Работа с диском экономит I/O(ps)
- Работа с диском требует многопоточности
- Многопоточность требует блокировок





Что можно сделать особенного?

- Монопольный доступ к памяти (1 поток)
- Оптимизация под память
- Специализированные аллокаторы
- Ориентация на OLTP (K/V)





Выводы: зачем In-Memory?

Потому что нужна производительность!

Большинство решений даёт сотни тысяч OLTP с ядра.

Пример:

Tarantool развивает до 1M RPS смешанных операций



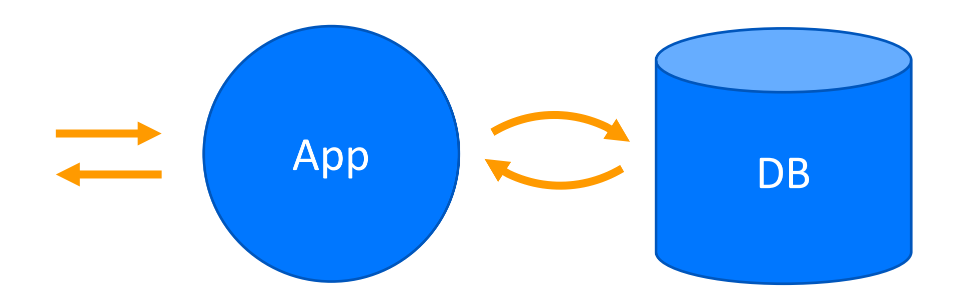


За счёт чего?

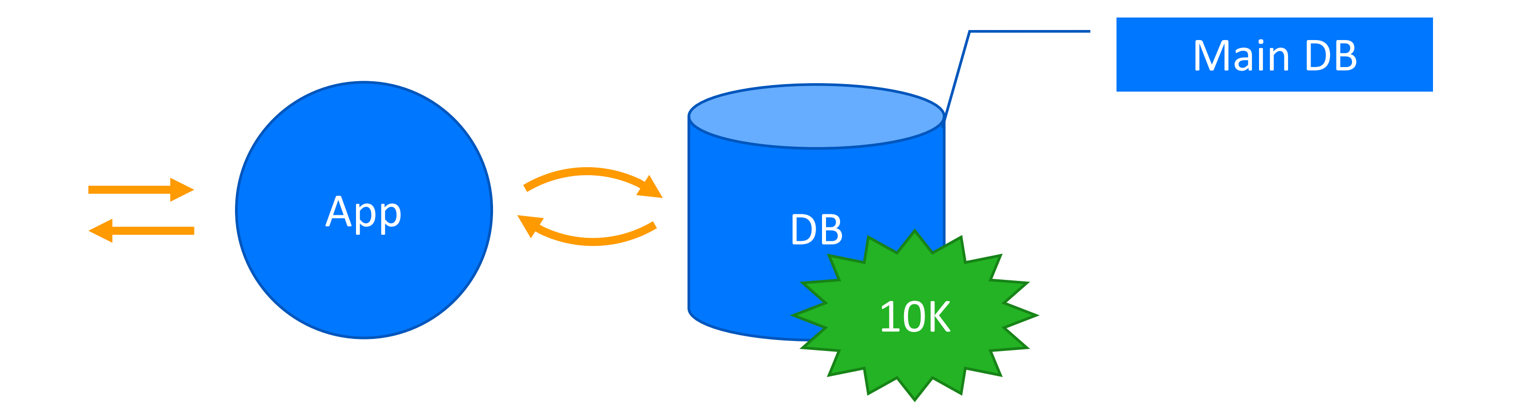
- Нет чтения диска
- Запись транзакций линейная
- Нет необходимости кешировать
- Идеология массового обслуживания



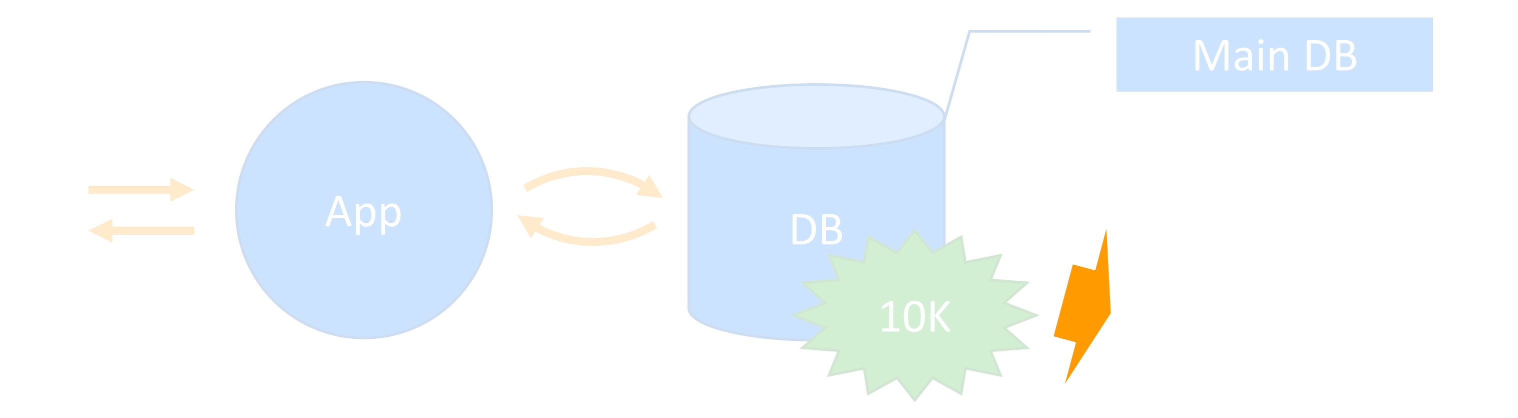
Выводы: зачем In-Memory?



Выводы: зачем In-Memory?



Выводы: зачем In-Memory?







Выводы: зачем In-Memory?

Main DB

App

DB

10K

App

RAM

DB

10K

Main DB







Выводы: зачем In-Memory?

Main DB

App

DB

10K

Cache

RAM

100K

App

DB

10K

Main DB

Выводы: зачем In-Memory?

Main DB

App

DB

10K

Cache

RAM

100K

App

DB

10K

Main DB

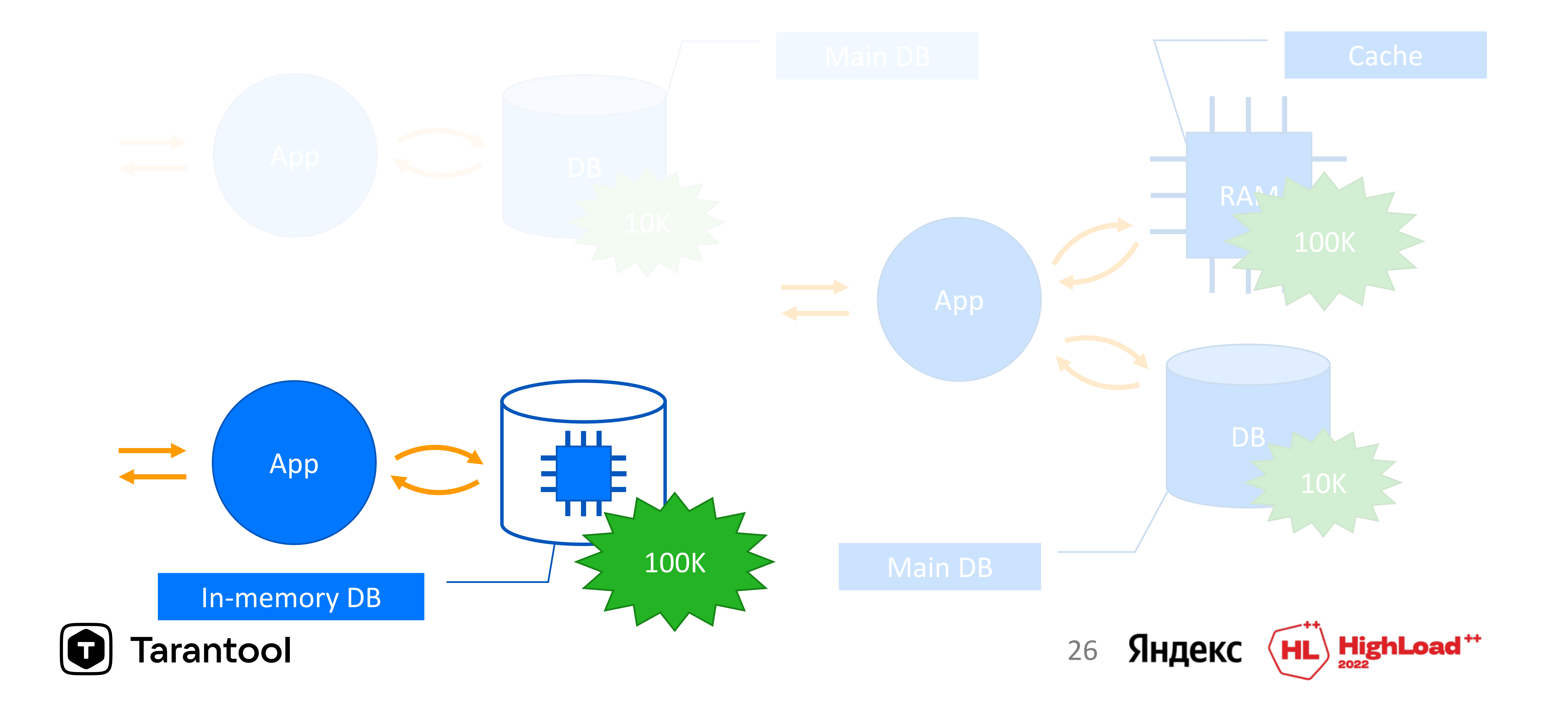
App







Выводы: зачем In-Memory?



Но какой ценой?






Did you do it?
Yes.
What did it cost?
Everything.


Everything.
In RAM.

02

От кеша к БД

Что отличает In-Memory БД от кеша в памяти?

История Tarantool

База данных

Кеш со вторичными индексами

Очередь

Платформа in-memory вычислений

База данных и сервер приложений

Мемкеш на стероидах

Резидентная БД

Как Redis, но лучше

Сервер приложений Lua

История Tarantool

in-memory cache

История Tarantool

in-memory *persistent* cache

История Tarantool



in-memory persistent cache

2008

in-memory persistent cache with replication

История Tarantool



in-memory persistent cache

2008

in-memory persistent cache with replication

in-memory key/value database

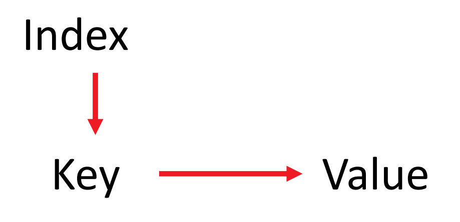
2010

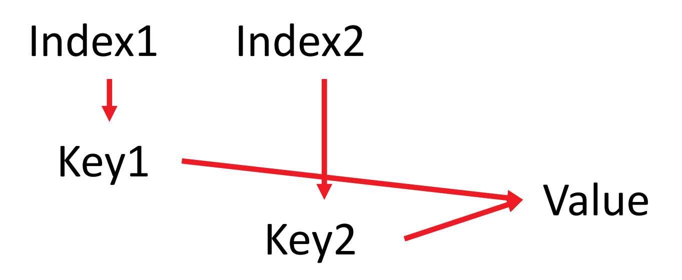
in-memory multi-index database (ACID)

Key/value to Multi-index

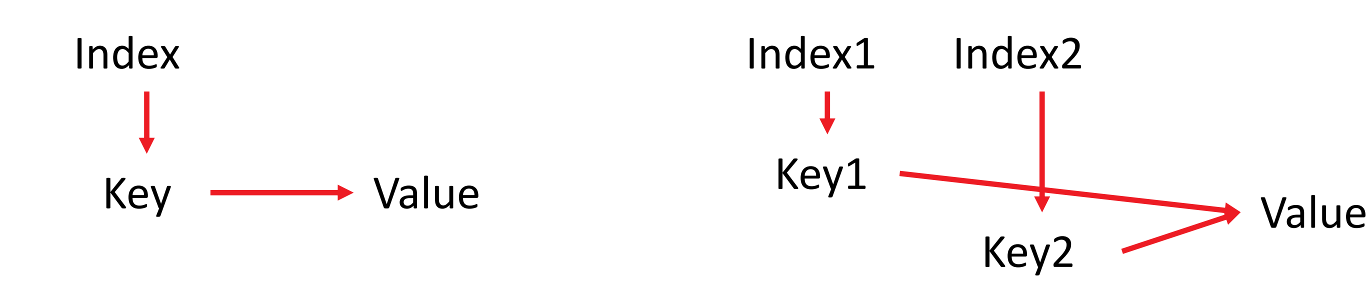


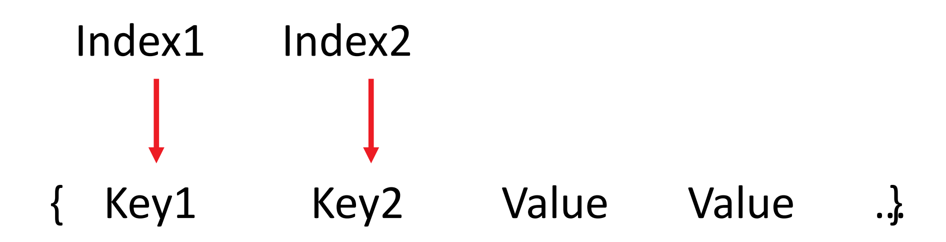
Key/value to Multi-index





Key/value to Multi-index





История Tarantool



in-memory persistent cache	2008
in-memory persistent cache with replication	
in-memory key/value database	2010
in-memory multi-index database (ACID)	
in-memory multi-index db with lua functions	2012

История Tarantool



in-memory persistent cache

in-memory persistent cache with replication — 2008

in-memory key/value database

in-memory multi-index database (ACID) — 2010

in-memory multi-index db with lua functions — 2012

in-memory multi-index db with cooperative runtime

История Tarantool



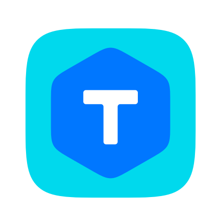
in-memory persistent cache	2008
in-memory persistent cache with replication	
in-memory key/value database	2010
in-memory multi-index database (ACID)	
in-memory multi-index db with lua functions	2012
in-memory multi-index db with cooperative runtime	
hybrid multi-index db with sql and lua app server	2016







История Tarantool



in-memory persistent cache

in-memory persistent cache with replication — 2008

in-memory key/value database

in-memory multi-index database (ACID) — 2010

in-memory multi-index db with lua functions — 2012

in-memory multi-index db with cooperative runtime

hybrid multi-index db with sql and lua app server * — 2016

* In-memory computing platform. With SQL/NoSQL In-memory database + Appliances

История Tarantool



in-memory persistent cache
in-memory persistent cache with replication — 2008
in-memory key/value database
in-memory multi-index database (ACID) — 2010
in-memory multi-index db with lua functions — 2012
in-memory multi-index db with cooperative runtime
hybrid multi-index db with sql and lua app server * — 2016







* In-memory computing platform. With SQL/NoSQL In-memory database + Appliances

ACID

Atomicity	Cooperative, 1 thread
Consistency	Coop monopoly of txn’s
Isolation	No parallel txn’s
Durability	Write Ahead Log





Cooperative vs Multithread

RAM: 0.5 M - 2.5 M чтений/s блоками по 4kb

1 чтение ~ 1 мкс => 1M RPS

Запись не требует блокировок

Основной код базы — не требует потокобезопасности



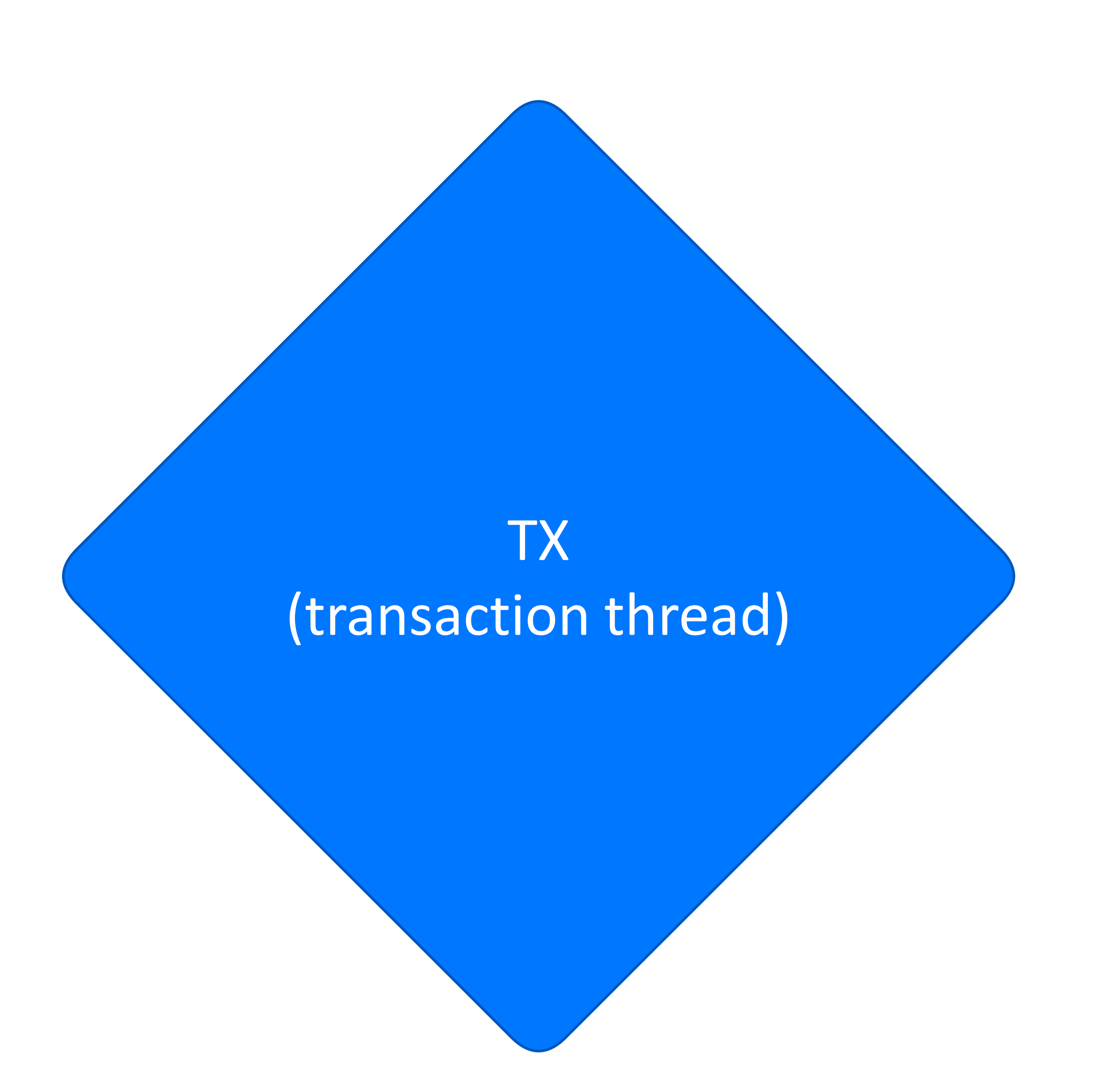


Как устроен Tarantool?

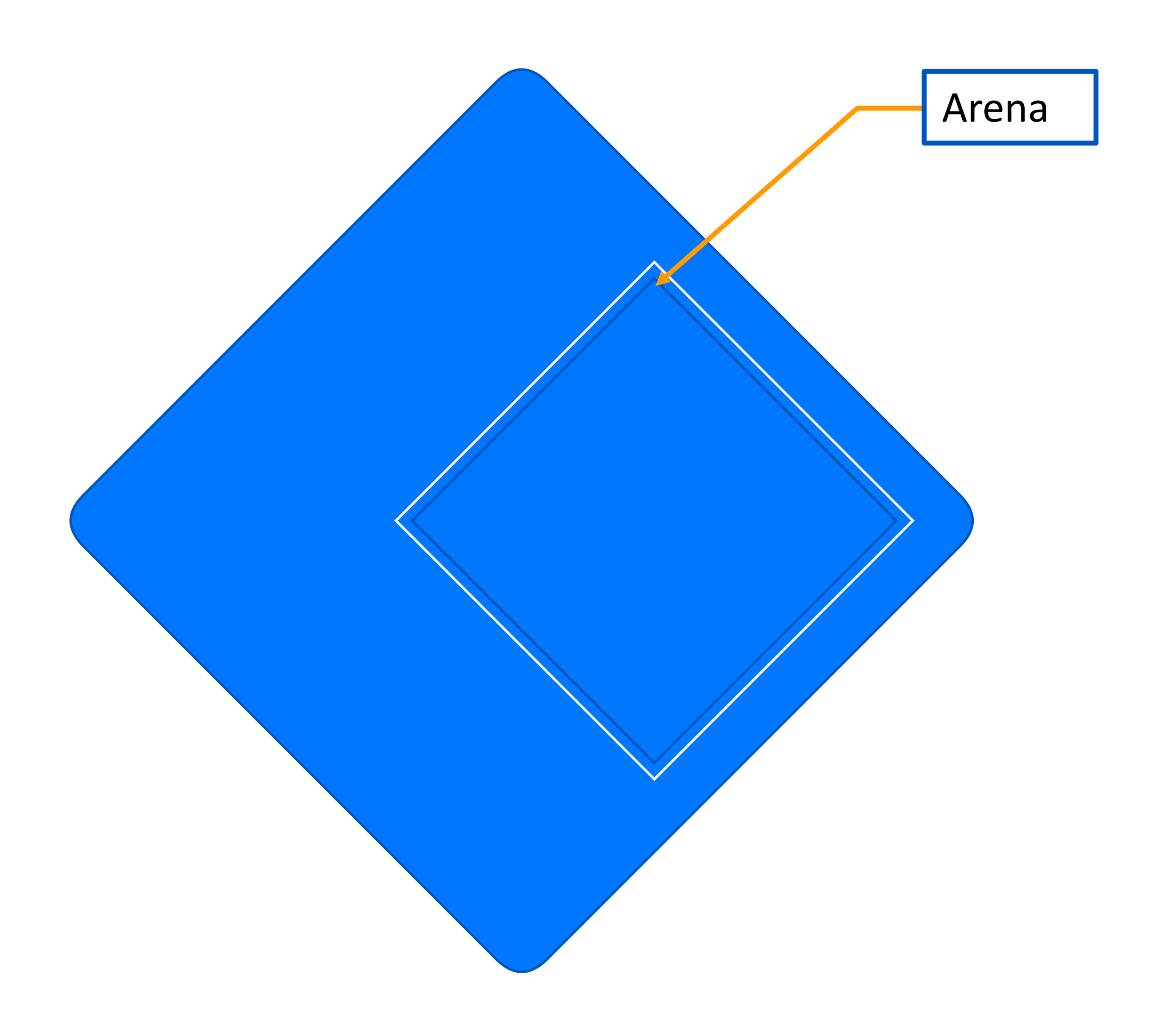
- 100% данных находятся в памяти
- Доступ к данным из одного потока
- Доступ к данным только через индекс
- Изменения пишутся во Write Ahead Log (WAL)
- WAL реплицируется
- Периодически сохранятся консистентный Snapshot



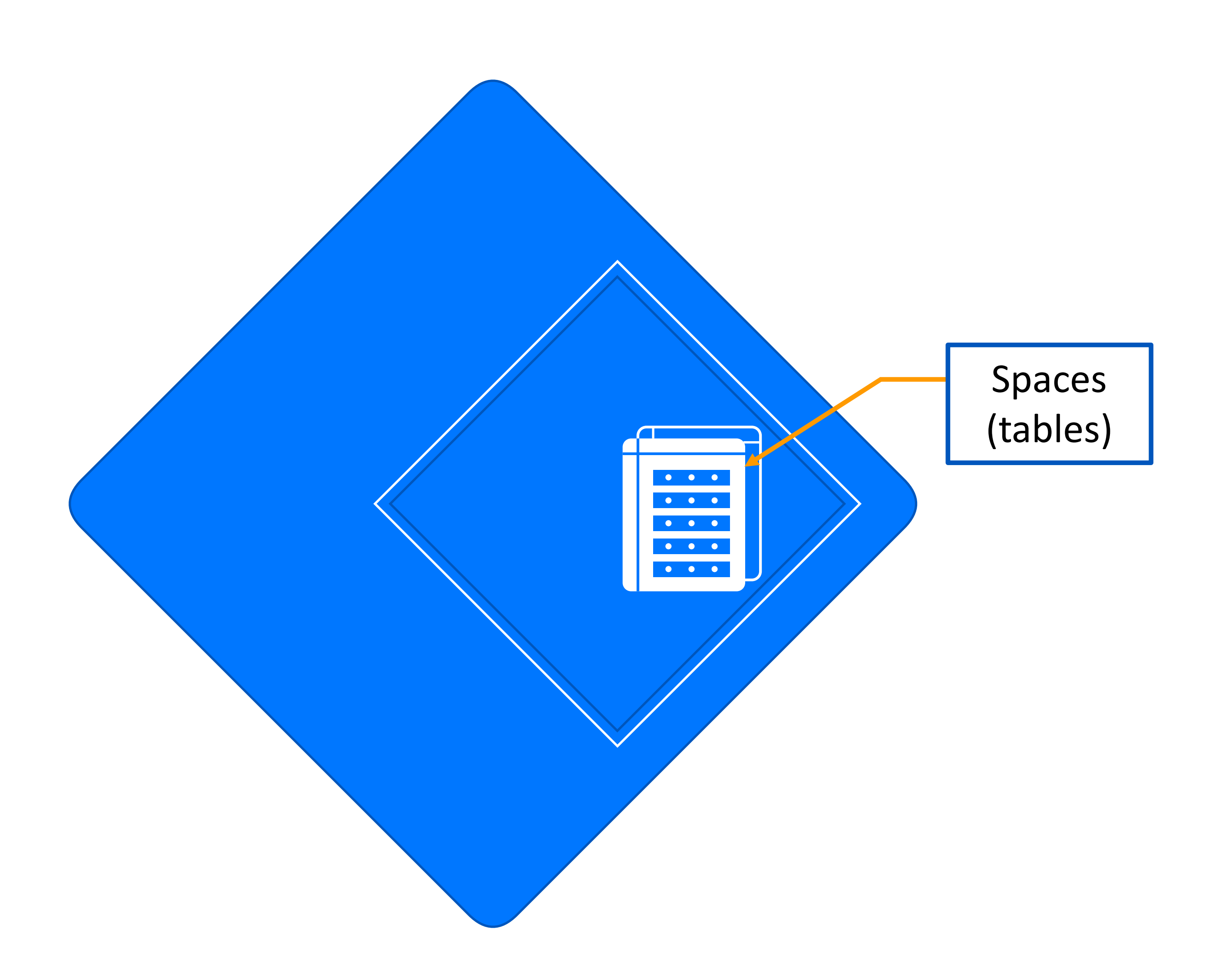




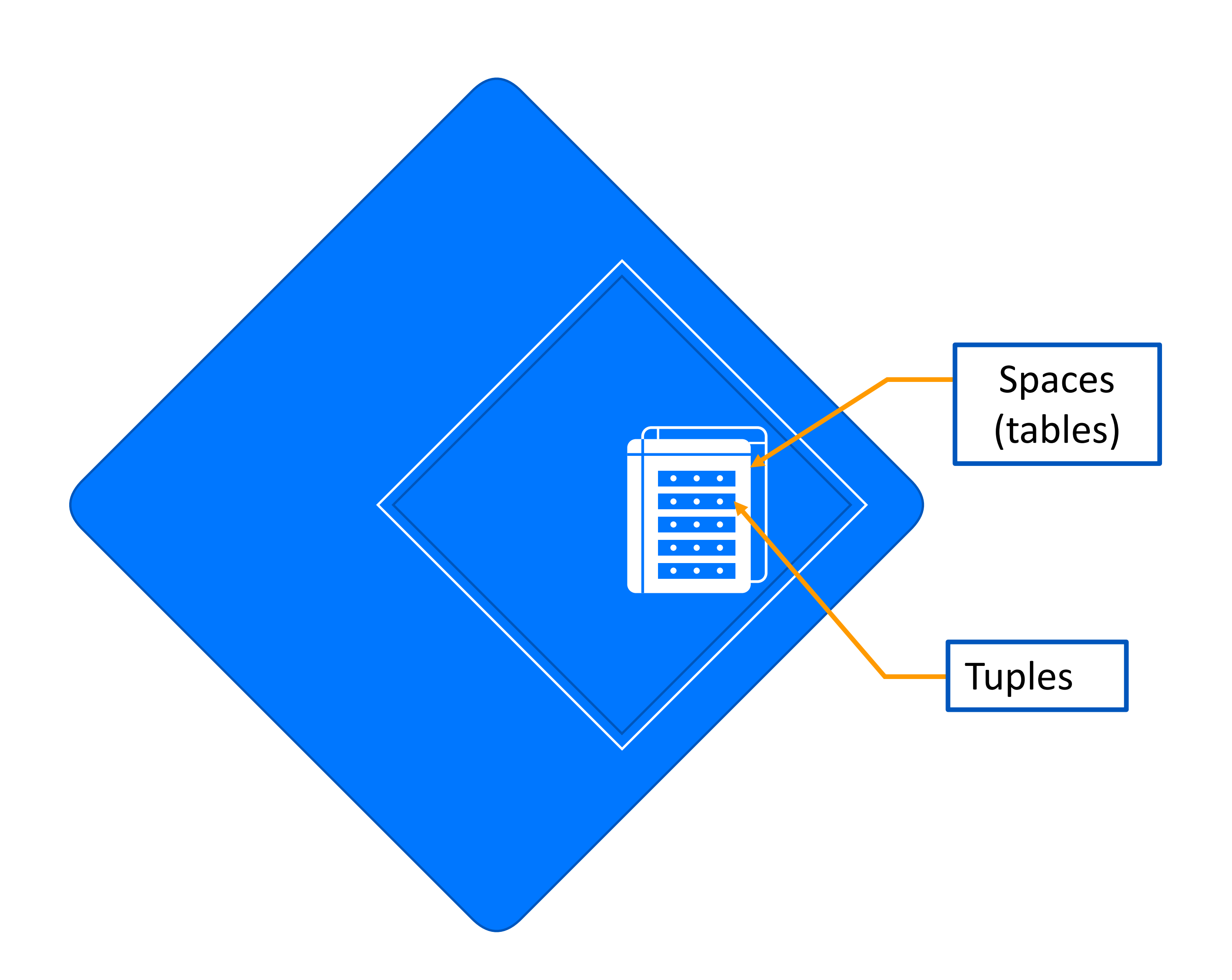
- Монопольный доступ



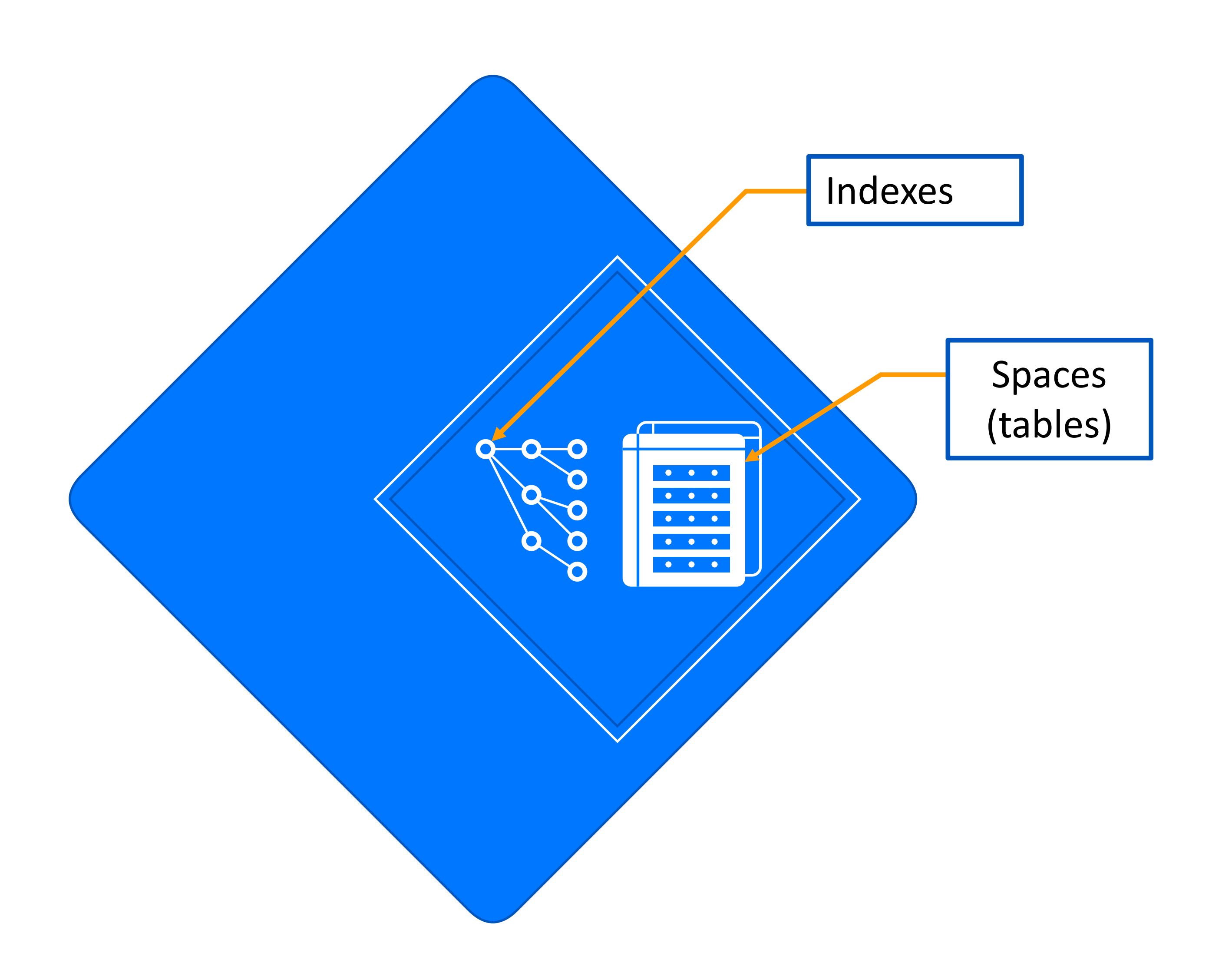
- Монопольный доступ
- Данные в памяти



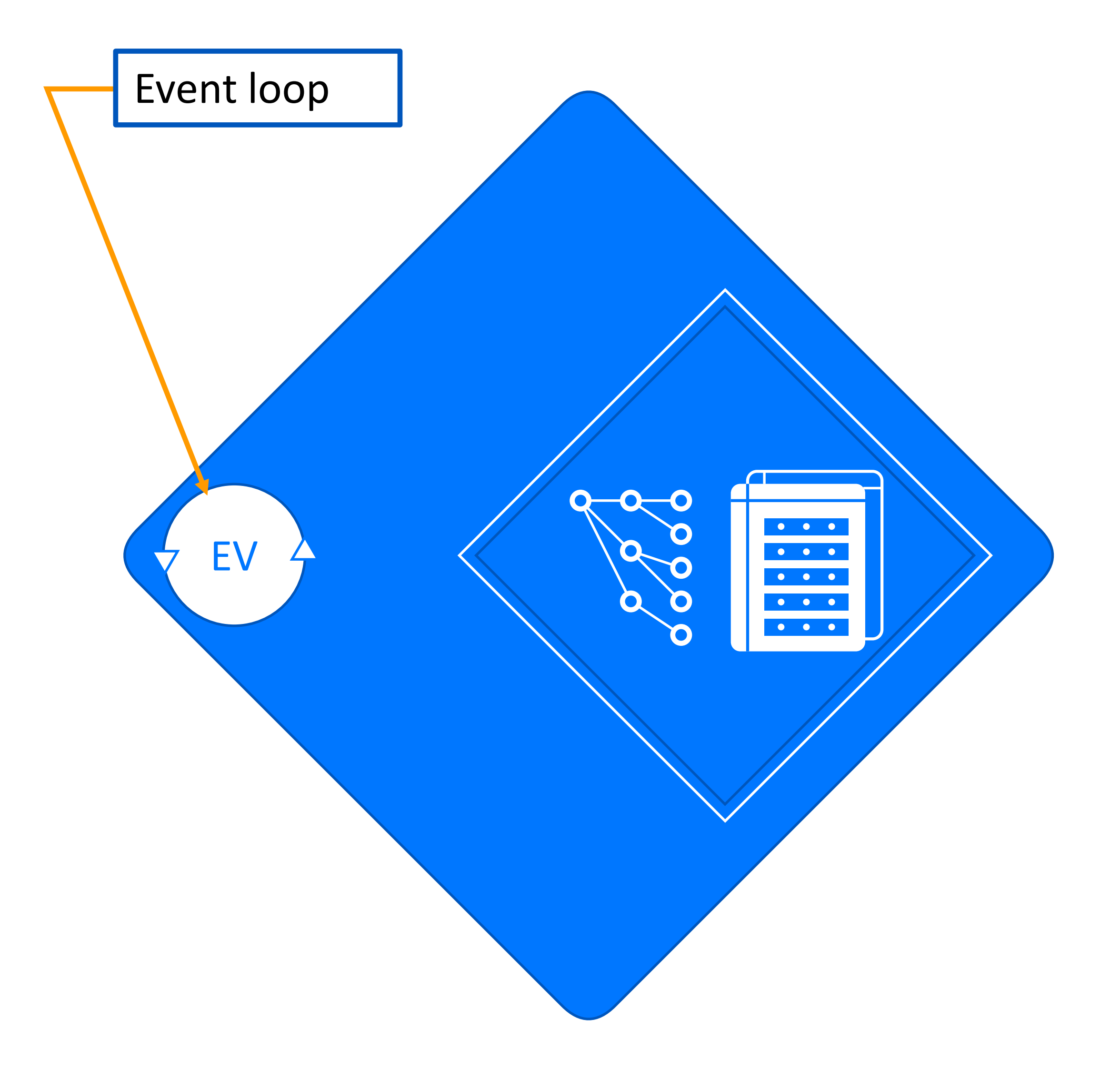
- Монопольный доступ
- Данные в памяти



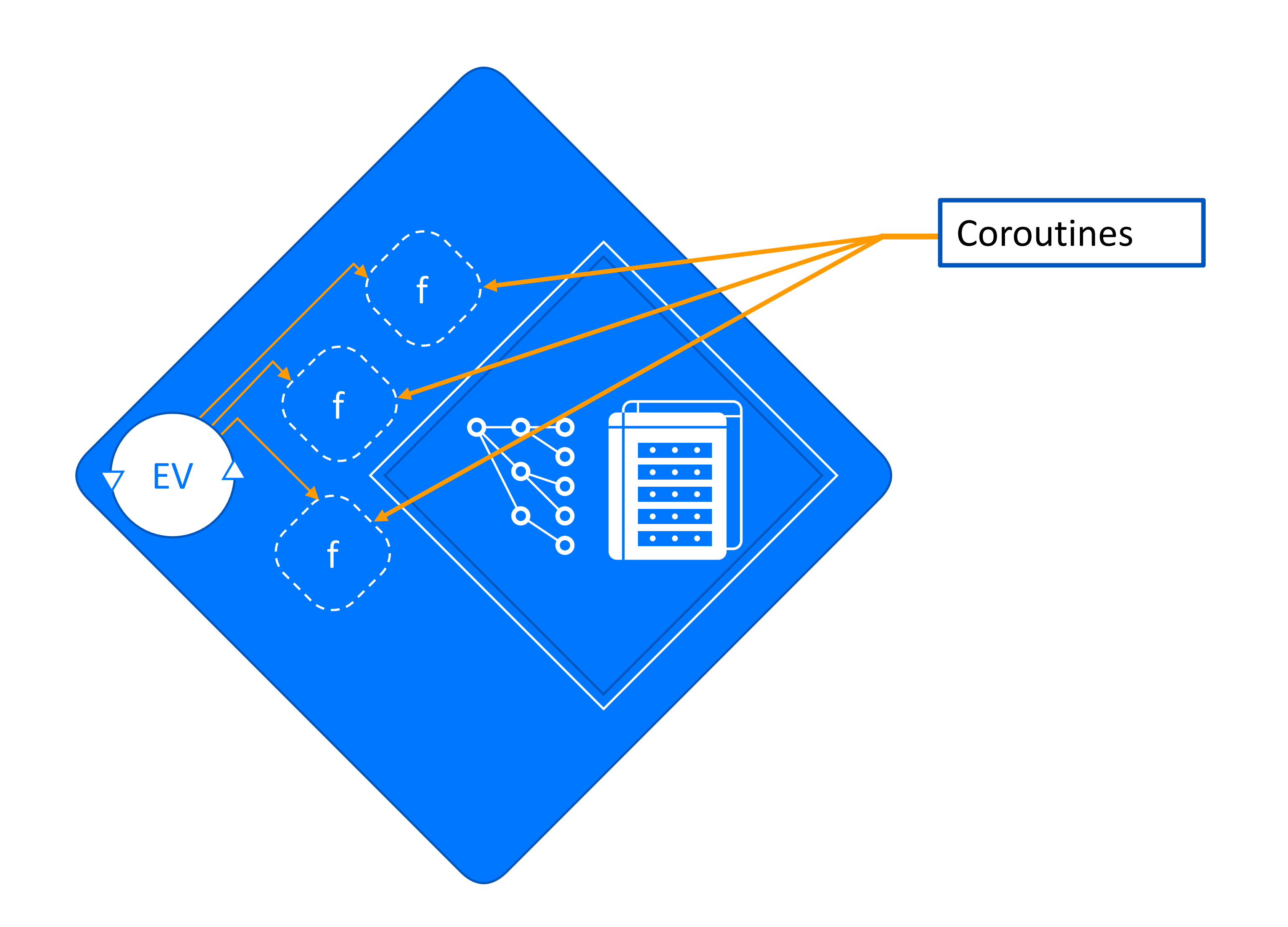
- Монопольный доступ
- Данные в памяти



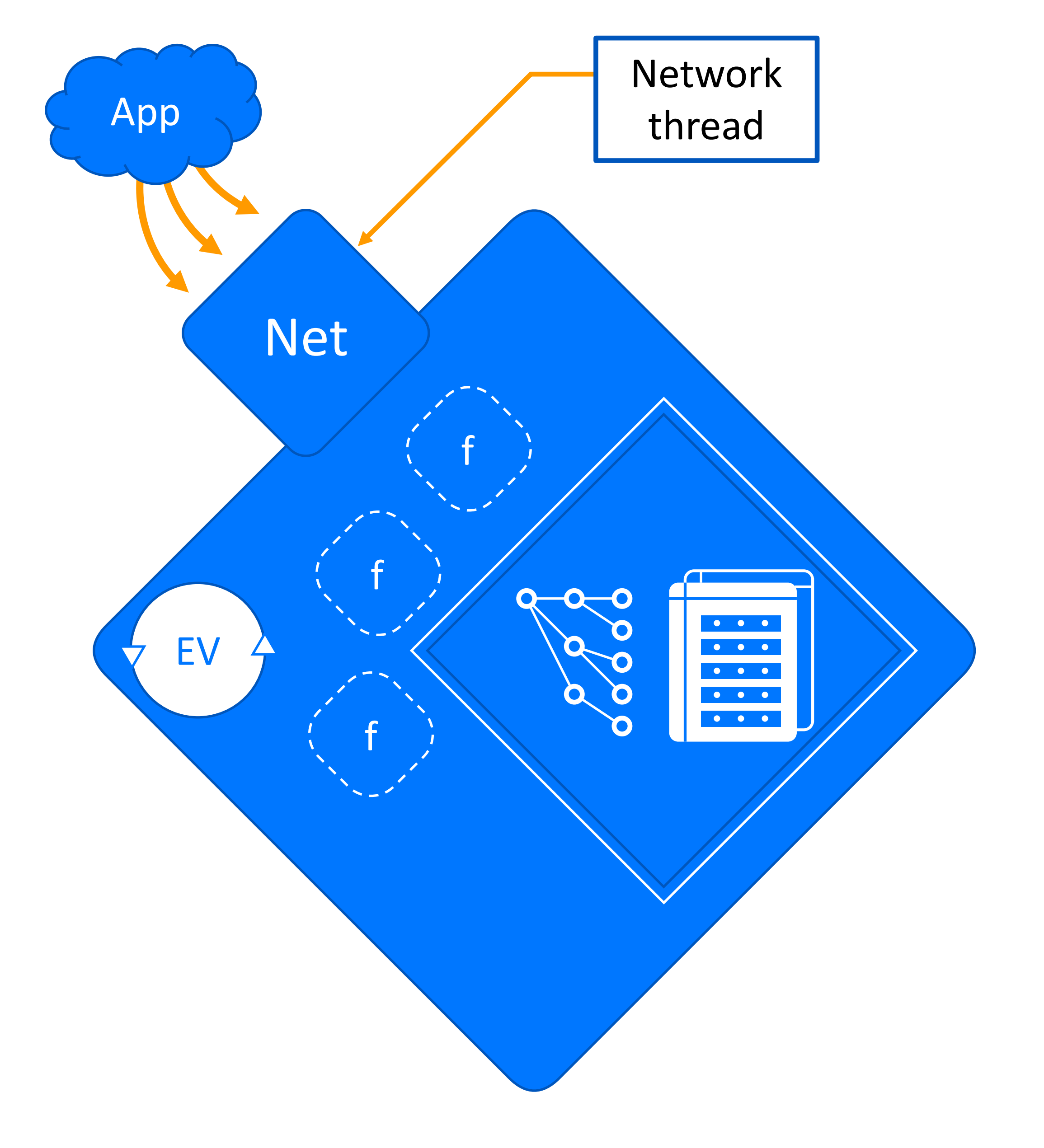
- Монопольный доступ
- Данные в памяти



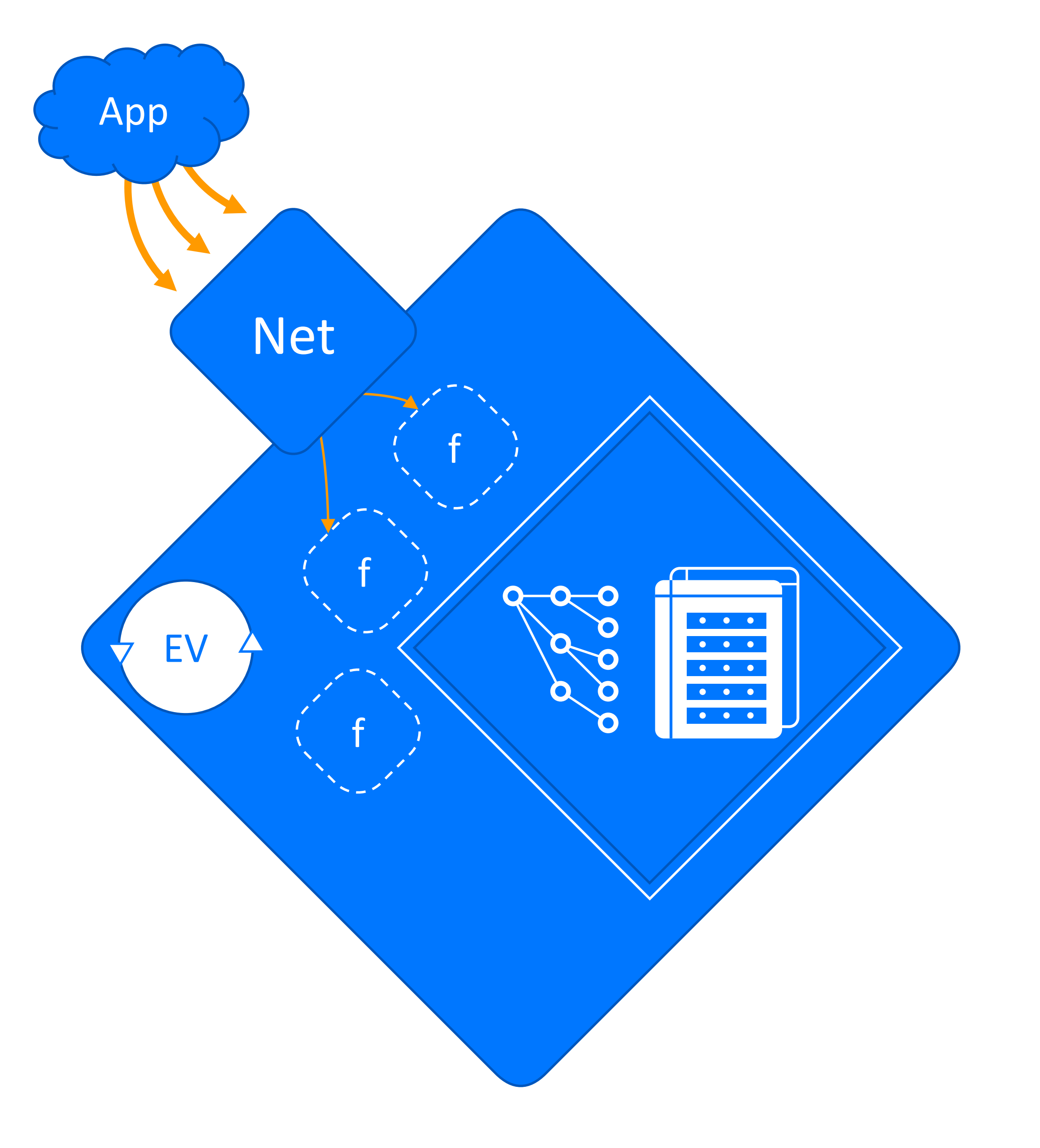
- Данные в памяти
- Монопольный доступ
- Событийный цикл



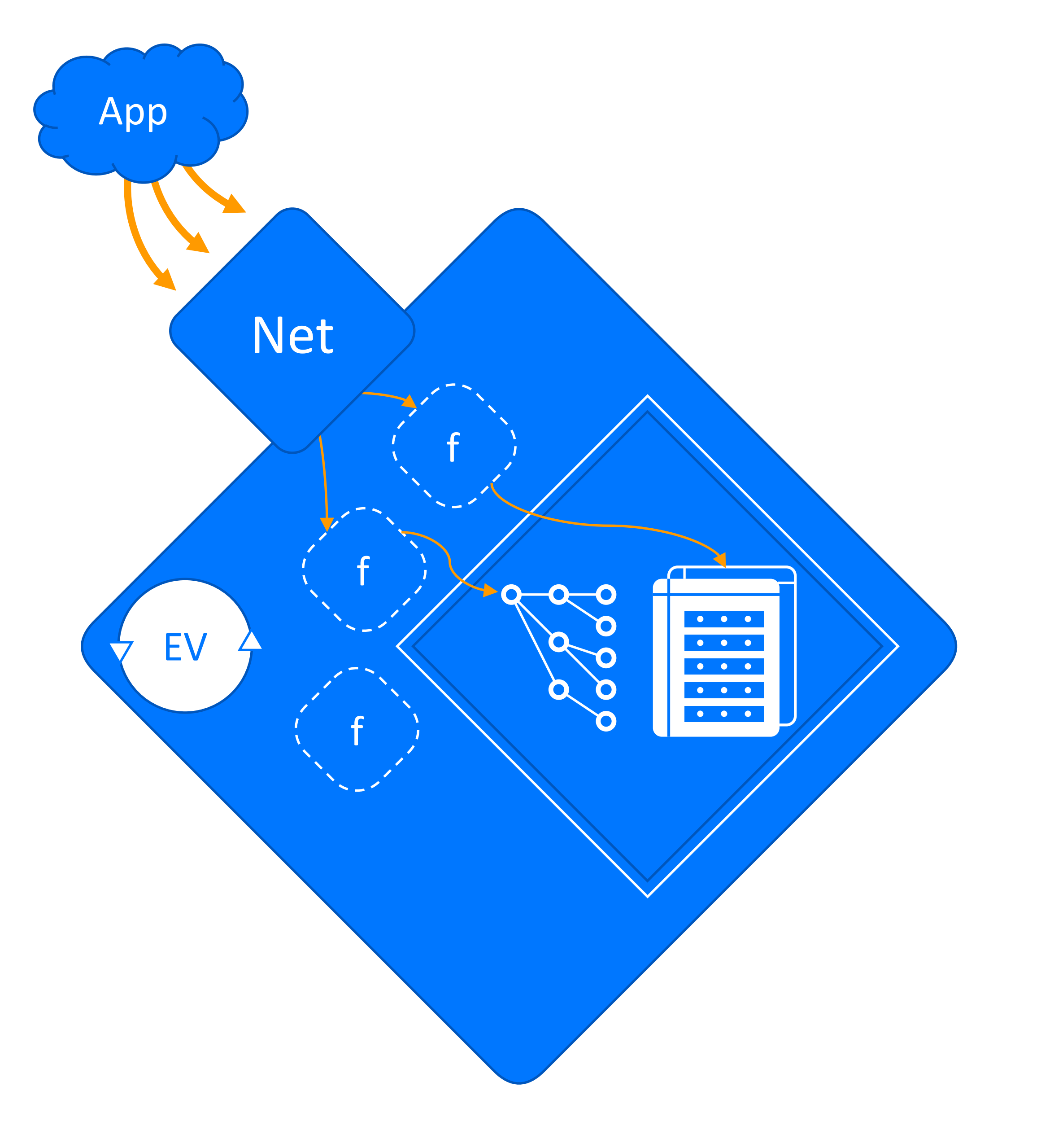
- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность



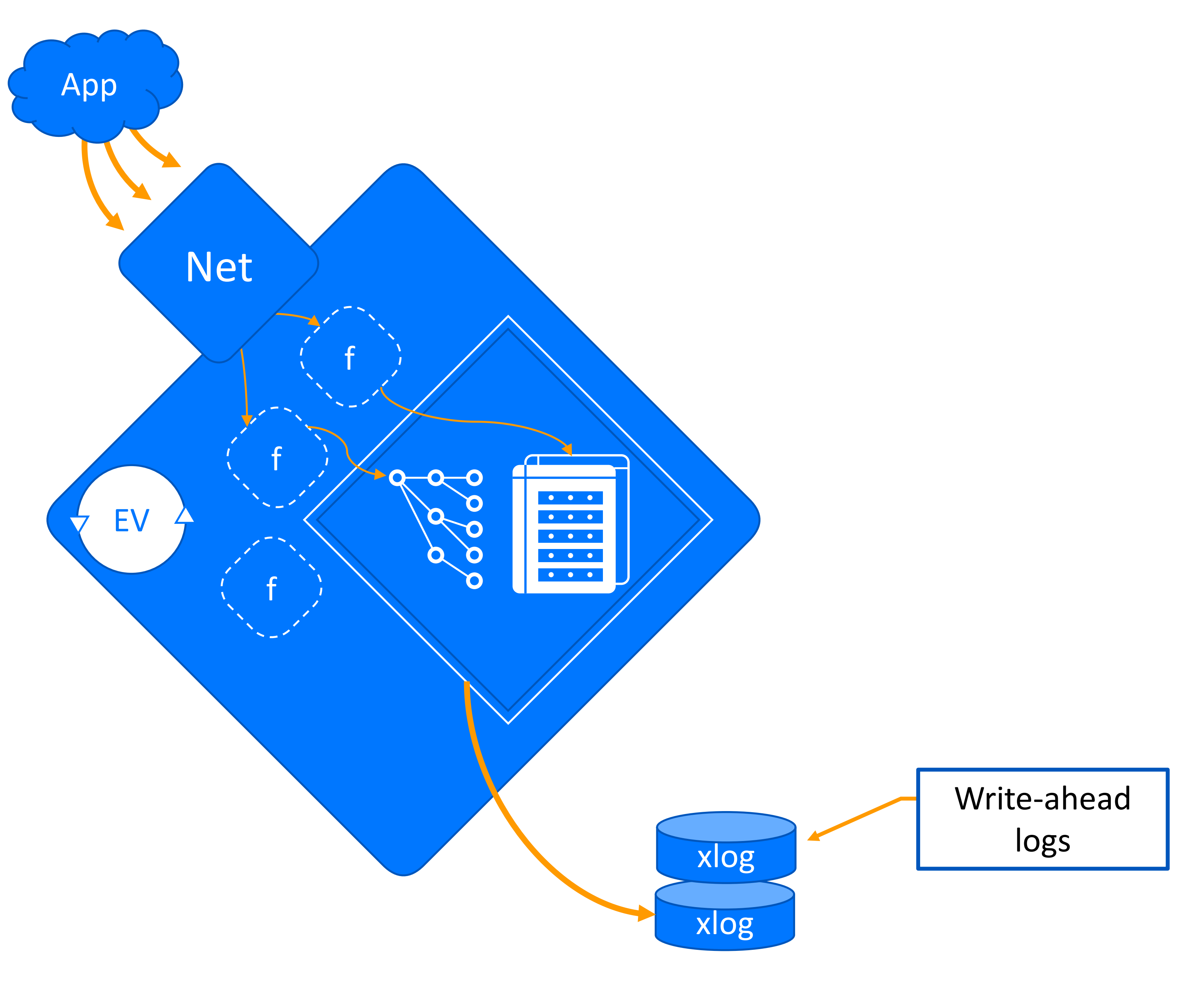
- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети



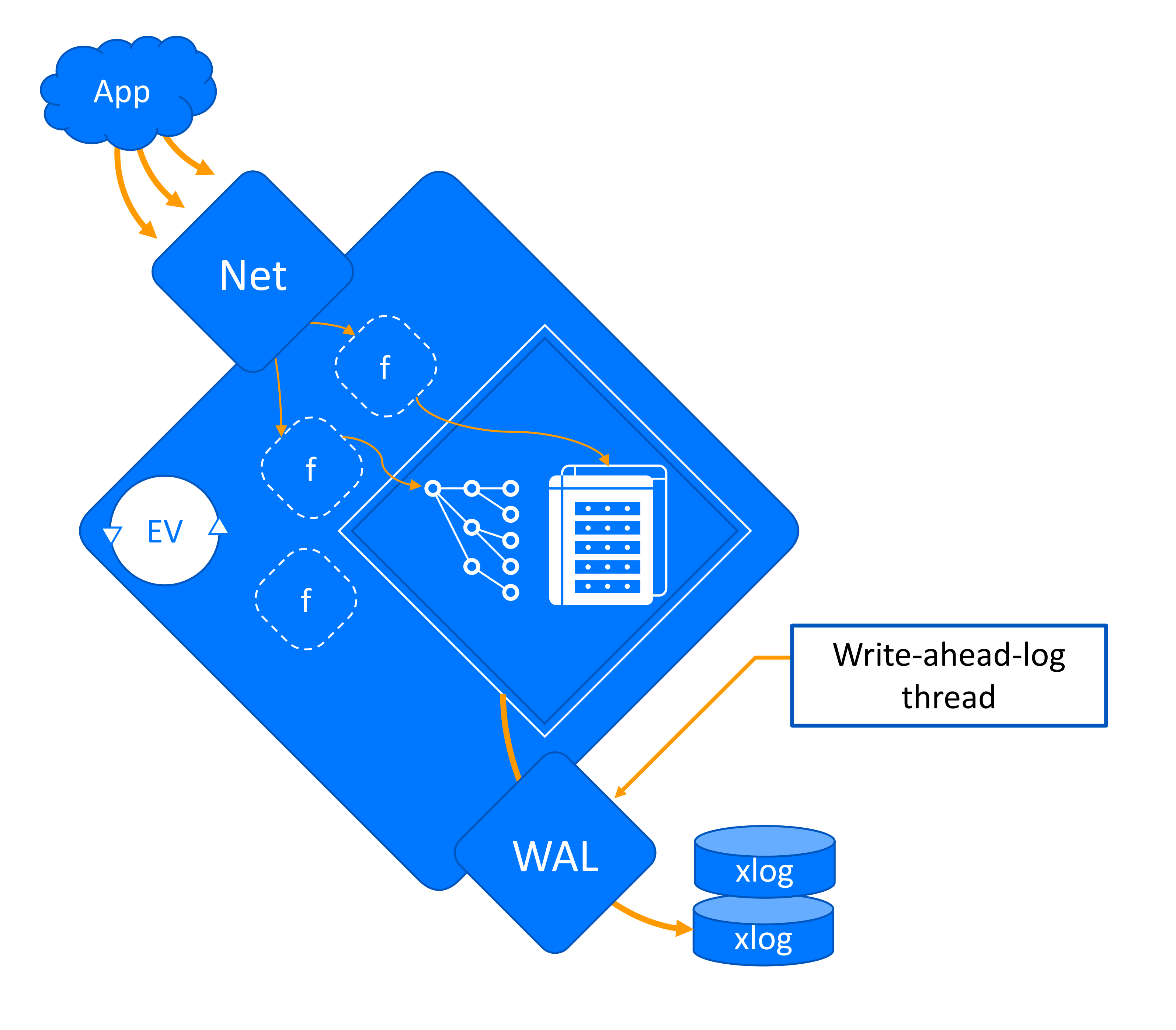
- Данные
  в памяти
- Монопольный
  доступ
- Событийный
  цикл
- Кооперативная
  многозадачность
- Отдельные
  потоки для сети



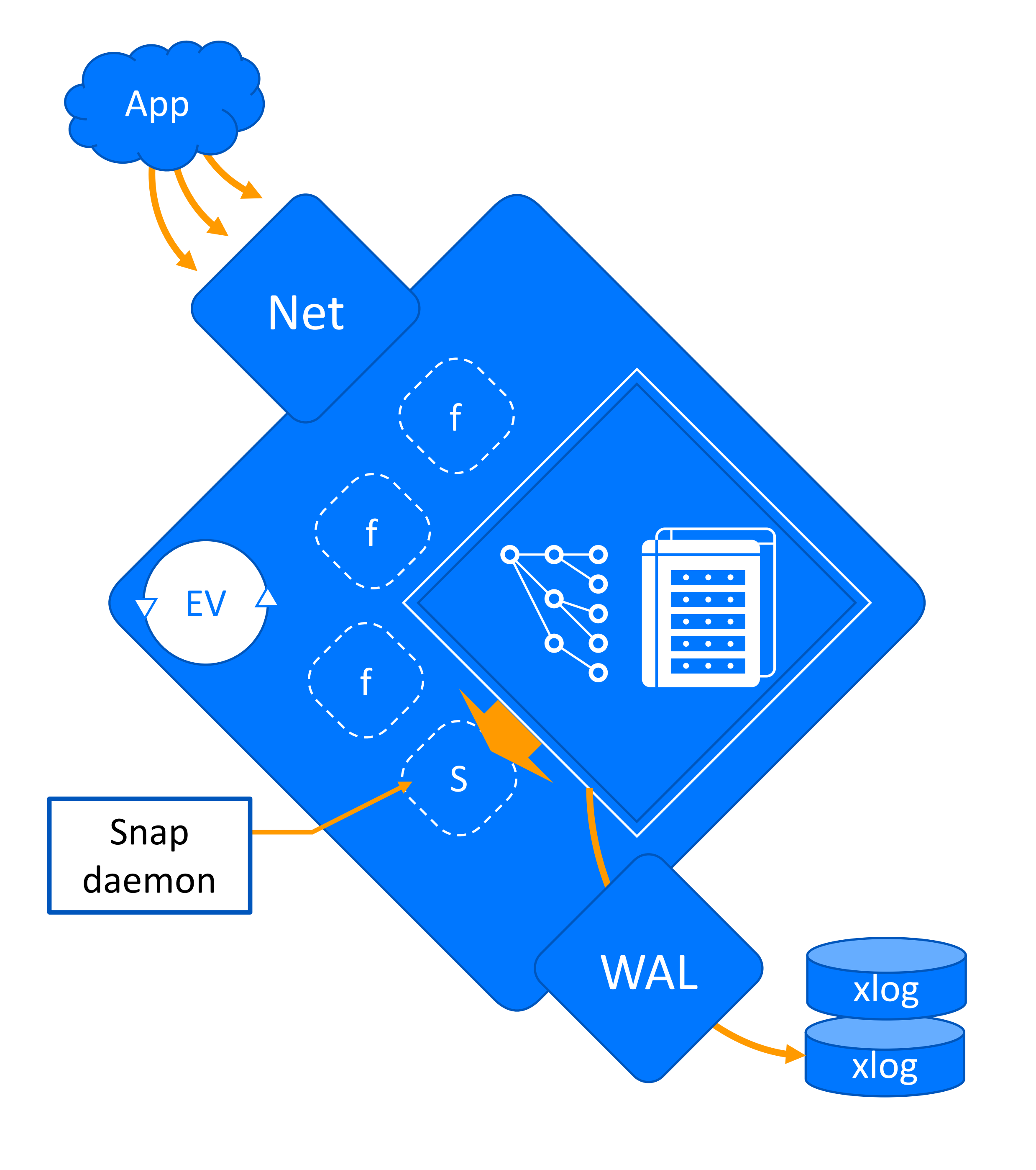
- Данные
  в памяти
- Монопольный
  доступ
- Событийный
  цикл
- Кооперативная
  многозадачность
- Отдельные
  потоки для сети



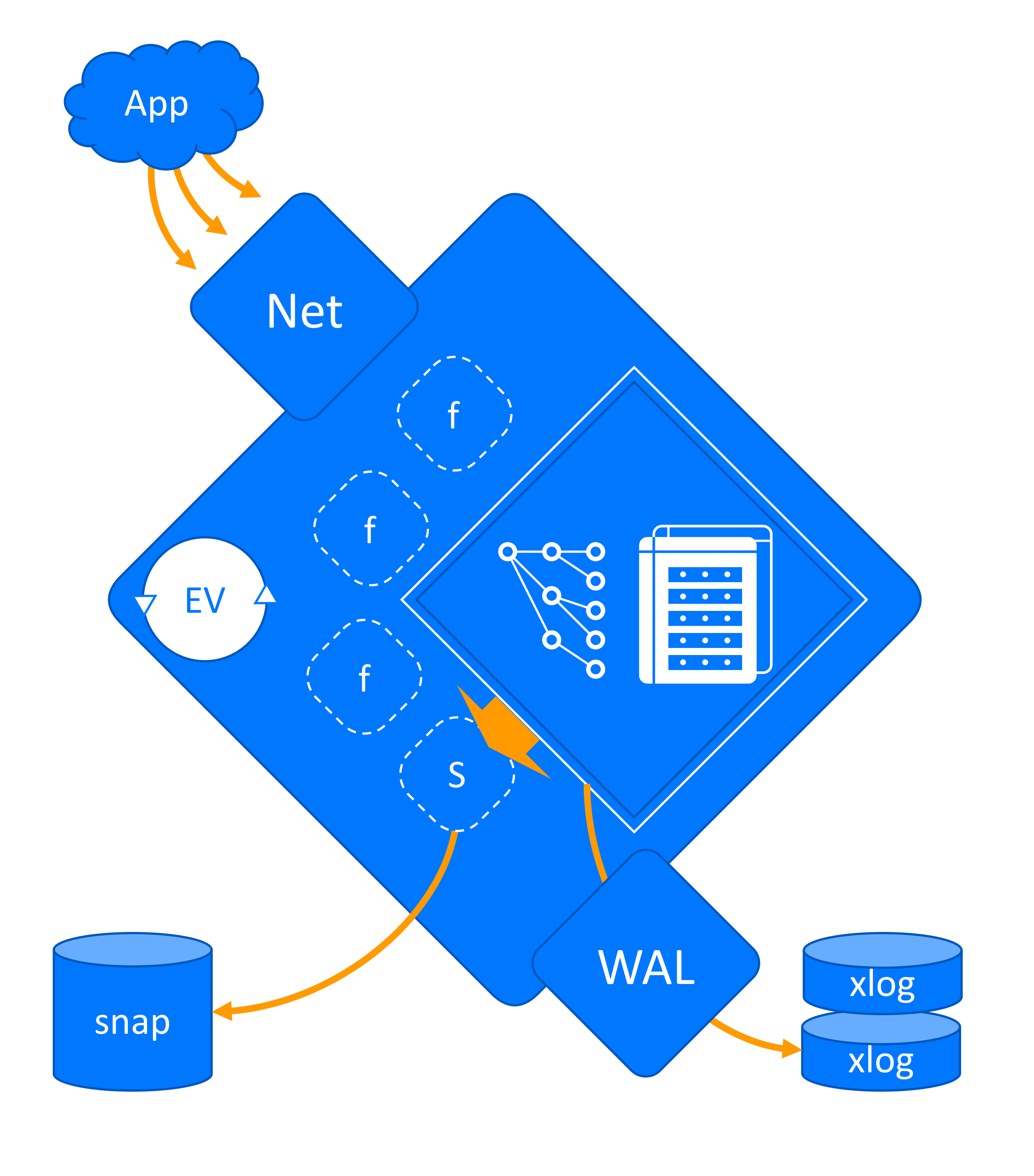
- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети
- Изменения во Write-Ahead-Log



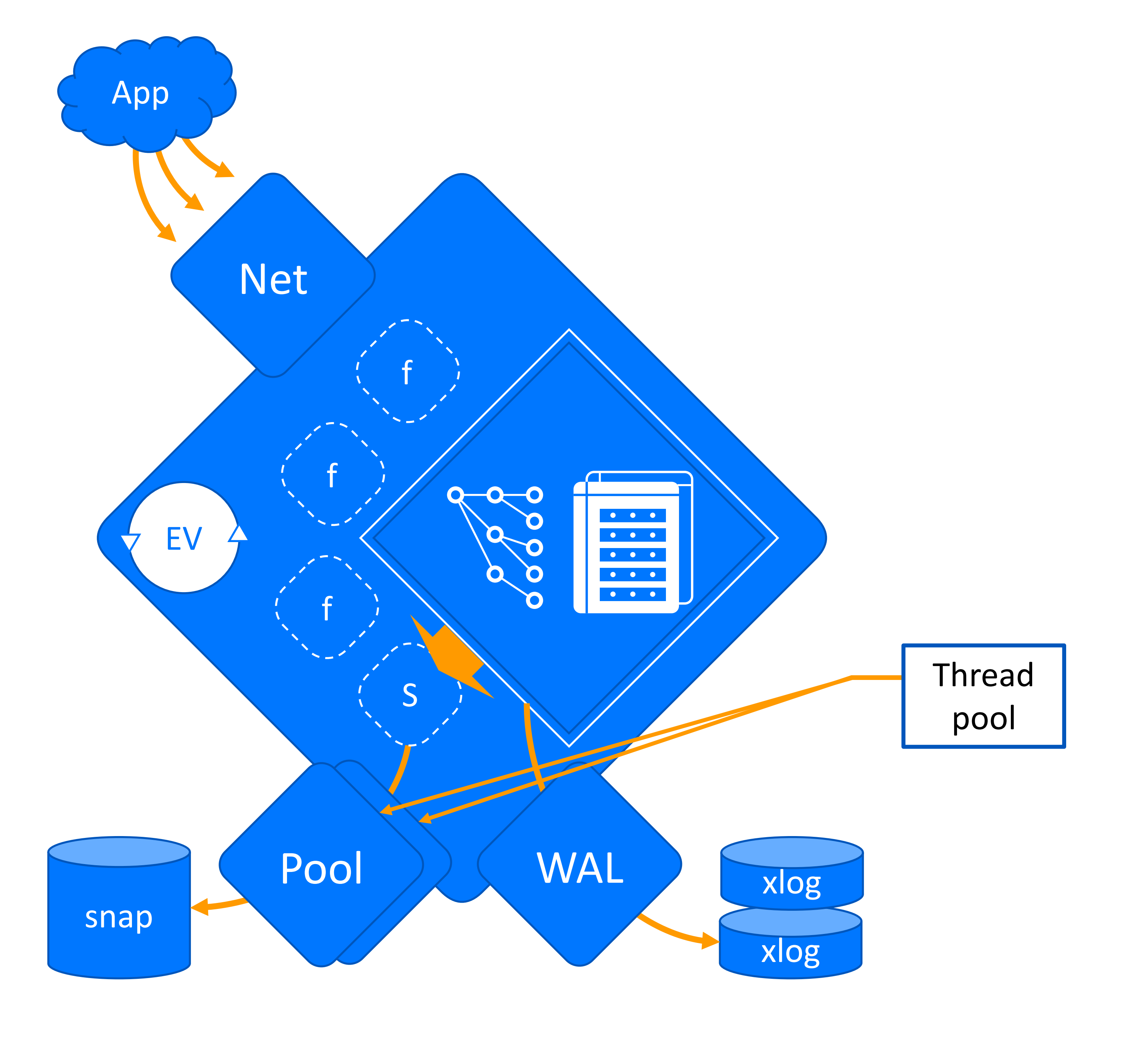
- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети
- Изменения во Write-Ahead-Log



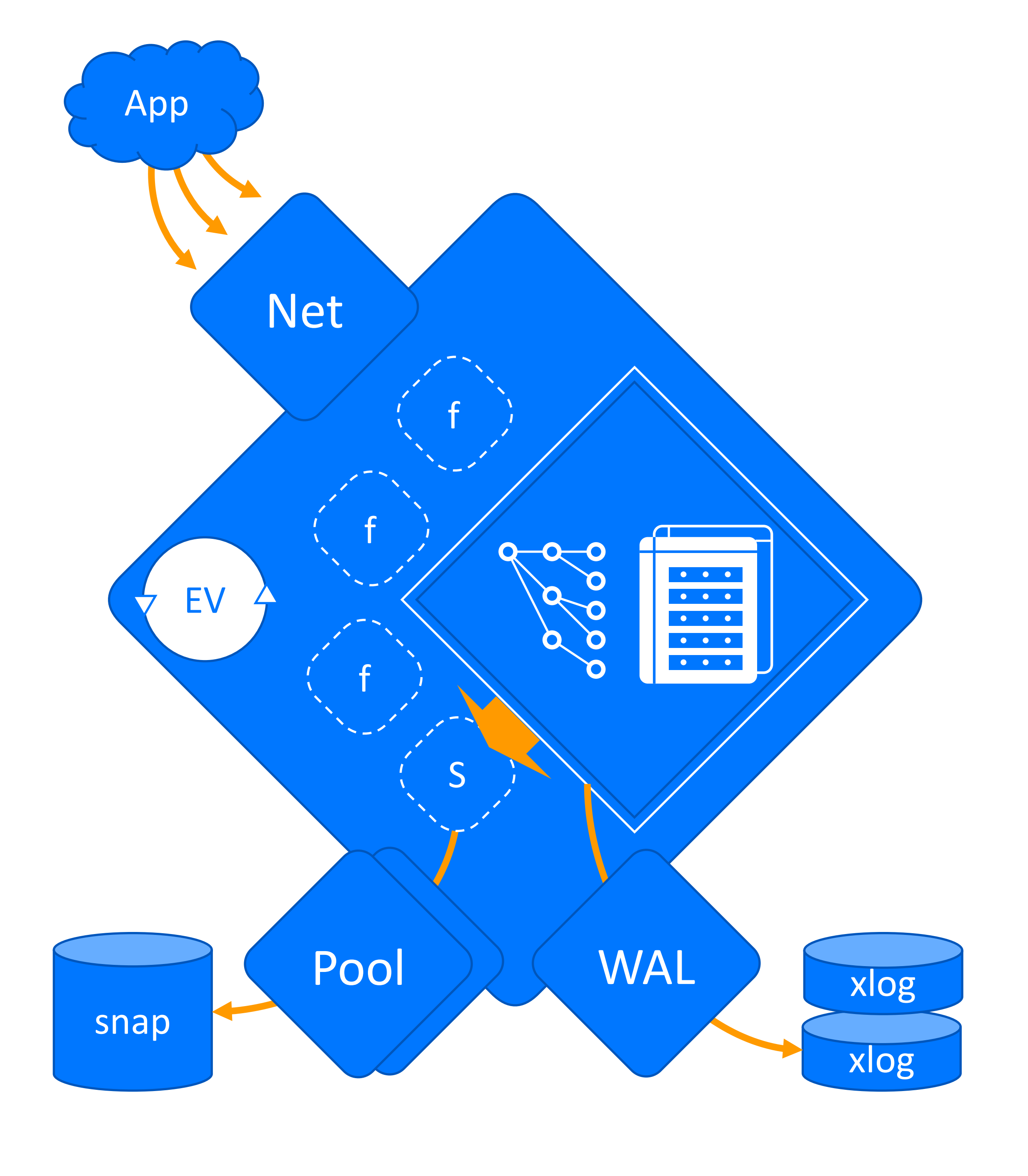
- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети
- Изменения во Write-Ahead-Log
- Консистентный снапшот



- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети
- Изменения во Write-Ahead-Log
- Консистентный снапшот



- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети
- Изменения во Write-Ahead-Log
- Консистентный снапшот
- Отдельные потоки для диска



- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети
- Изменения во Write-Ahead-Log
- Консистентный снапшот
- Отдельные потоки для диска

Откуда производительность?

- Диск медленный
- Системные вызовы дорогие





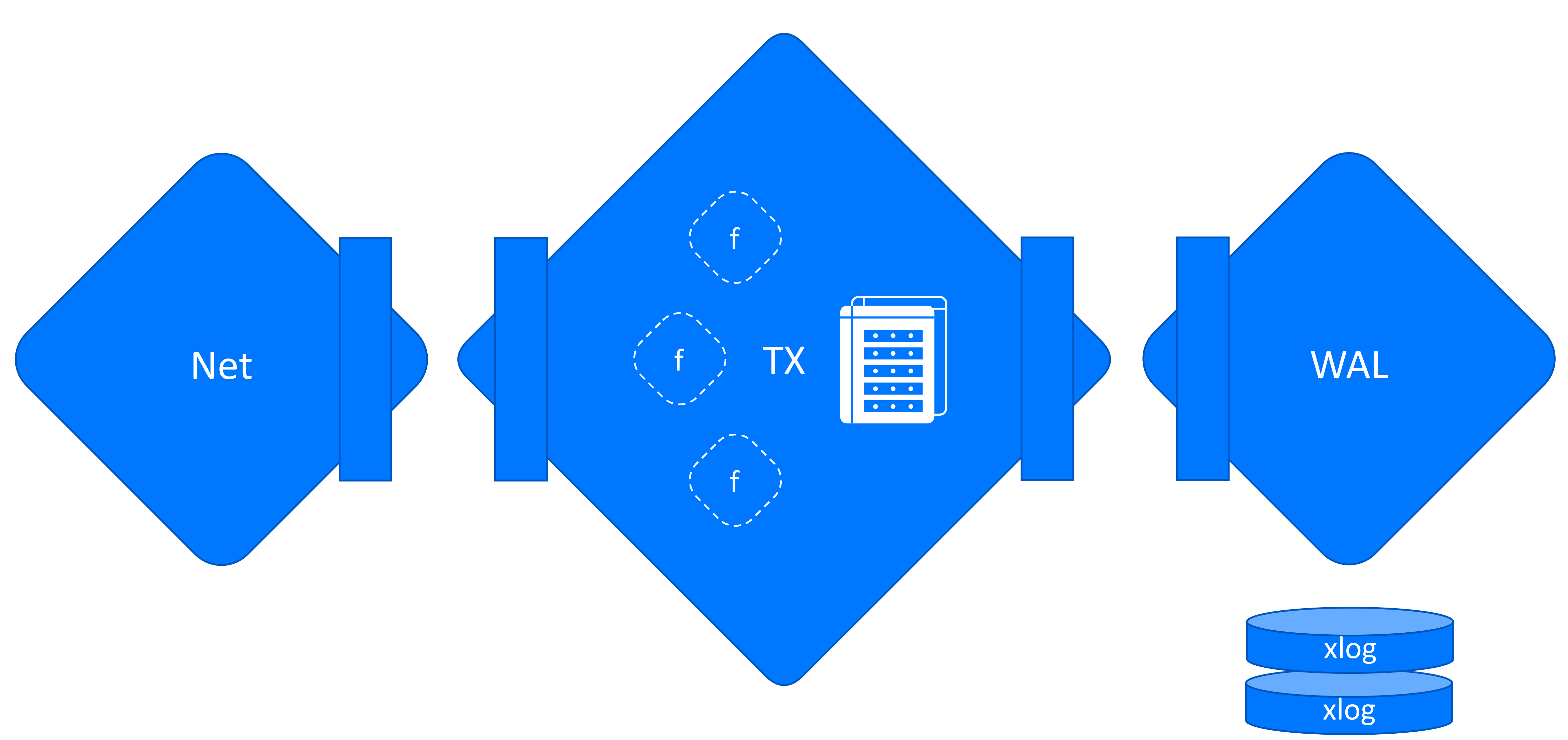
Откуда производительность?

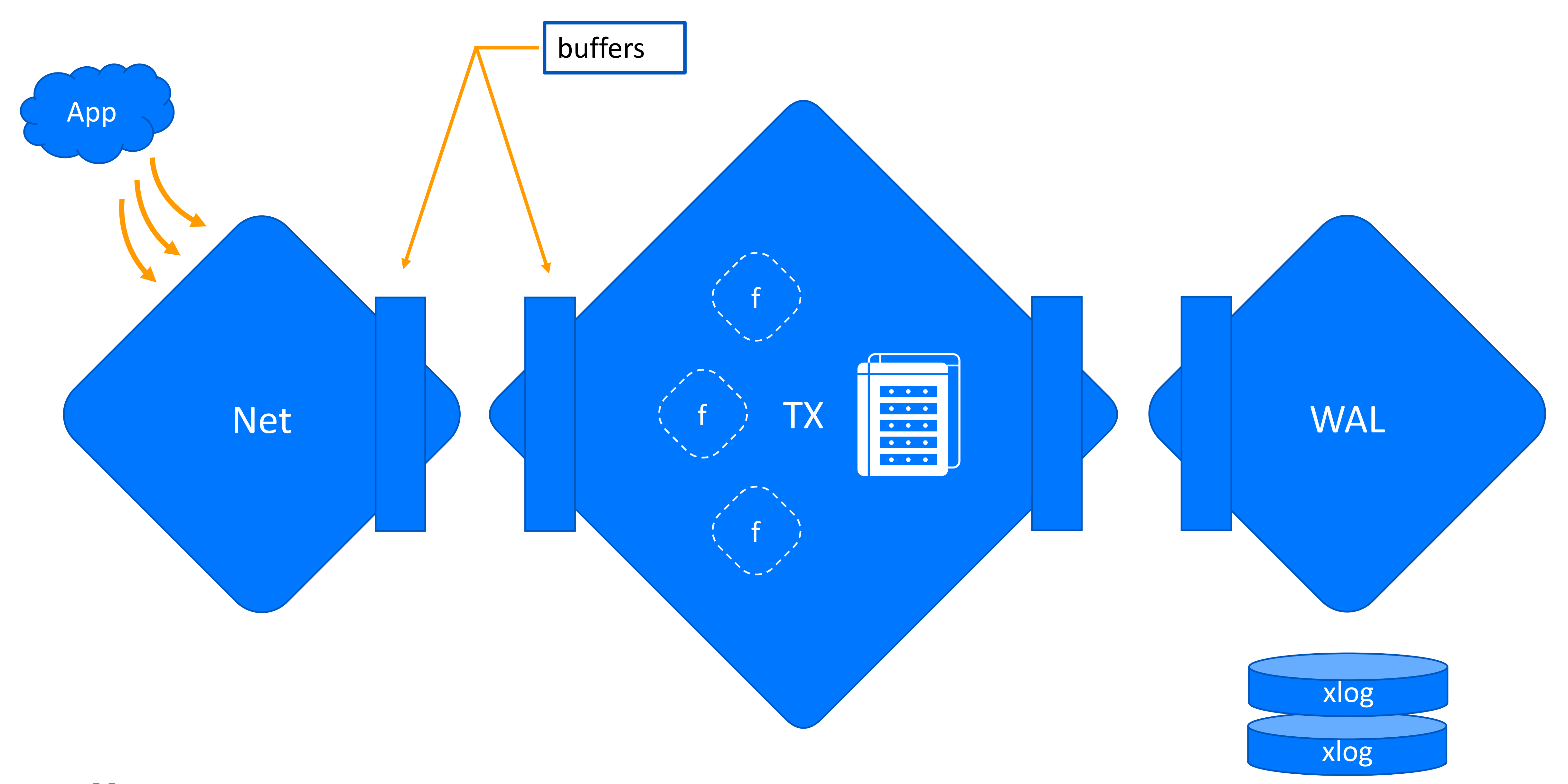
- Диск медленный
- Системные вызовы дорогие

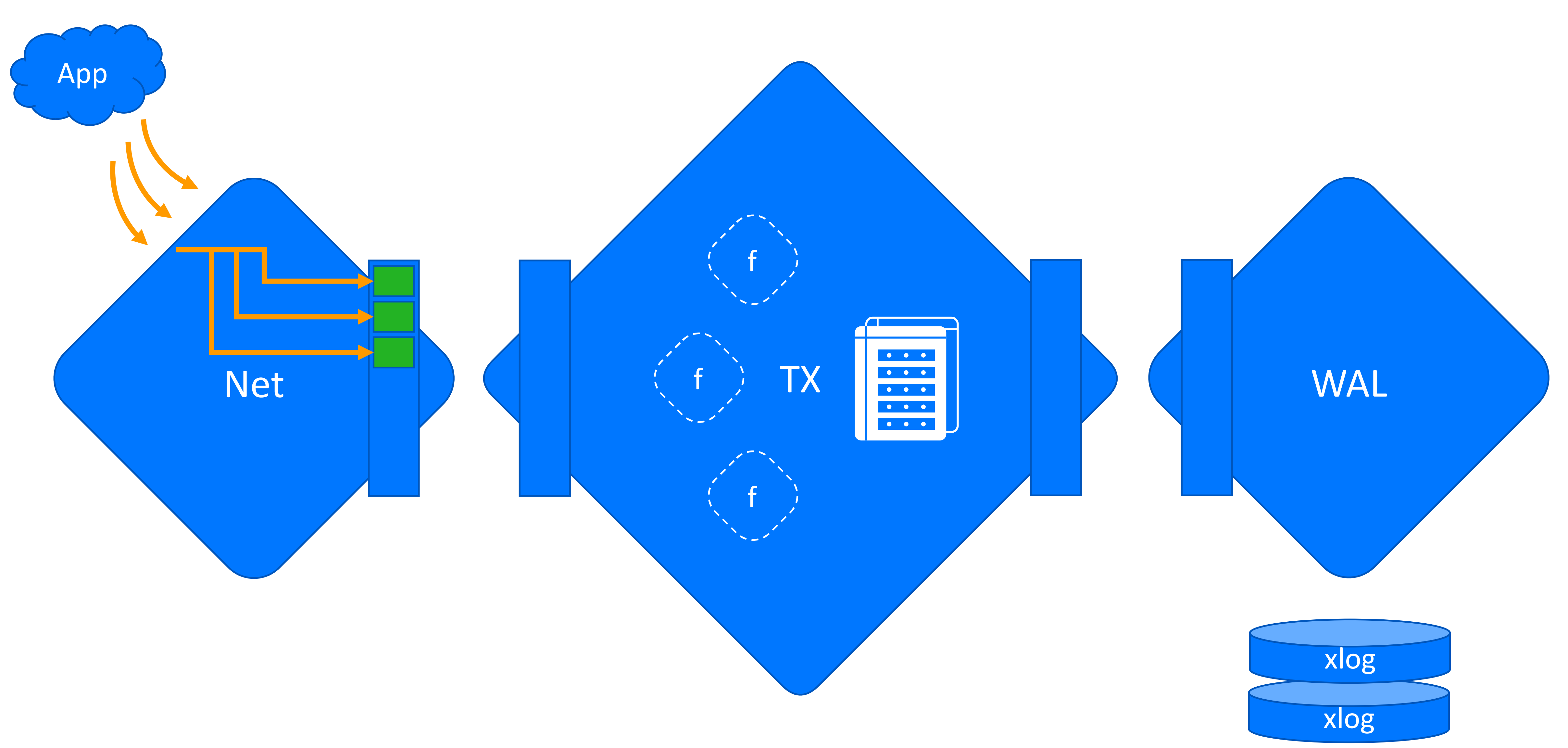
Батчинг!

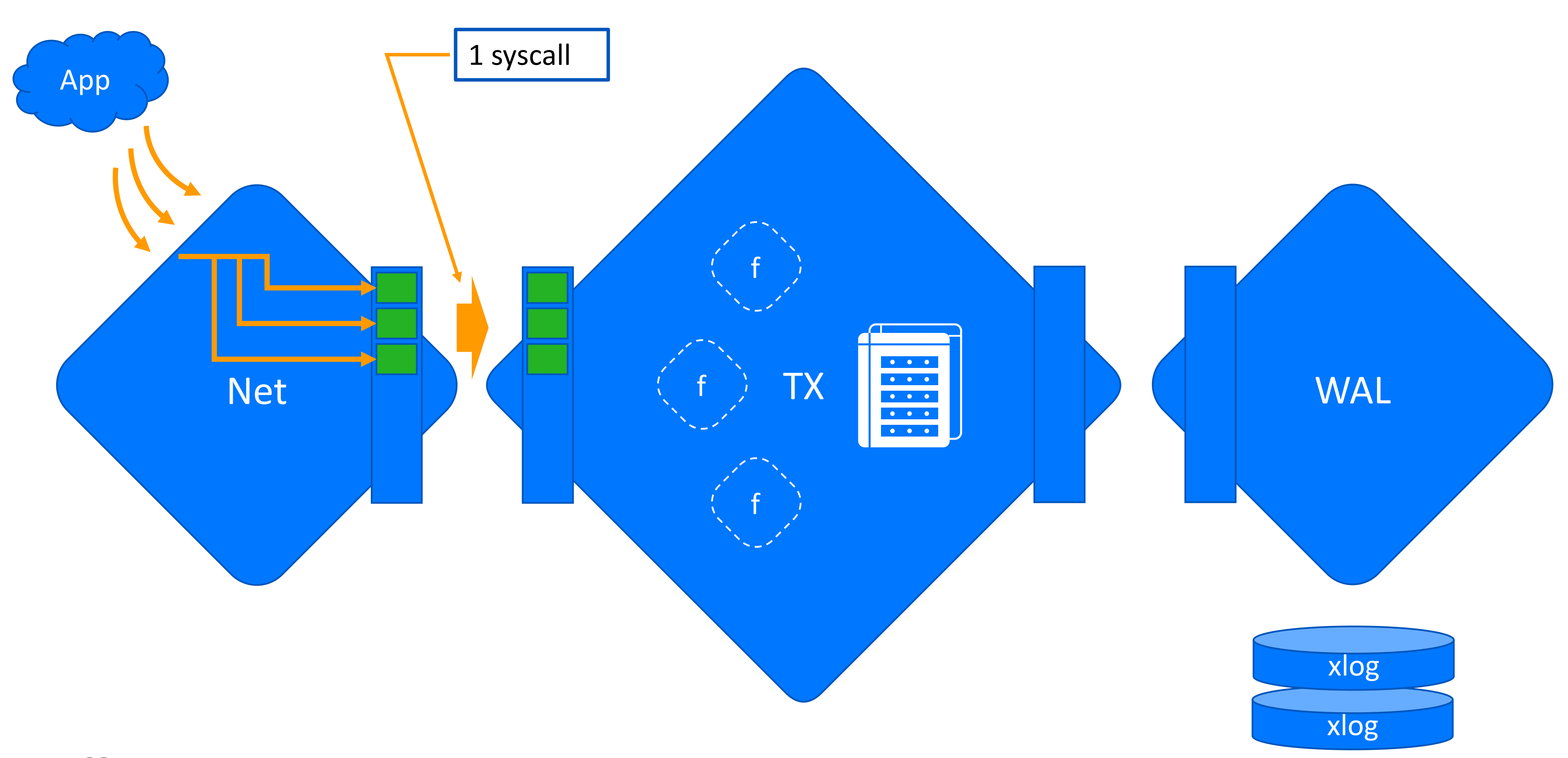




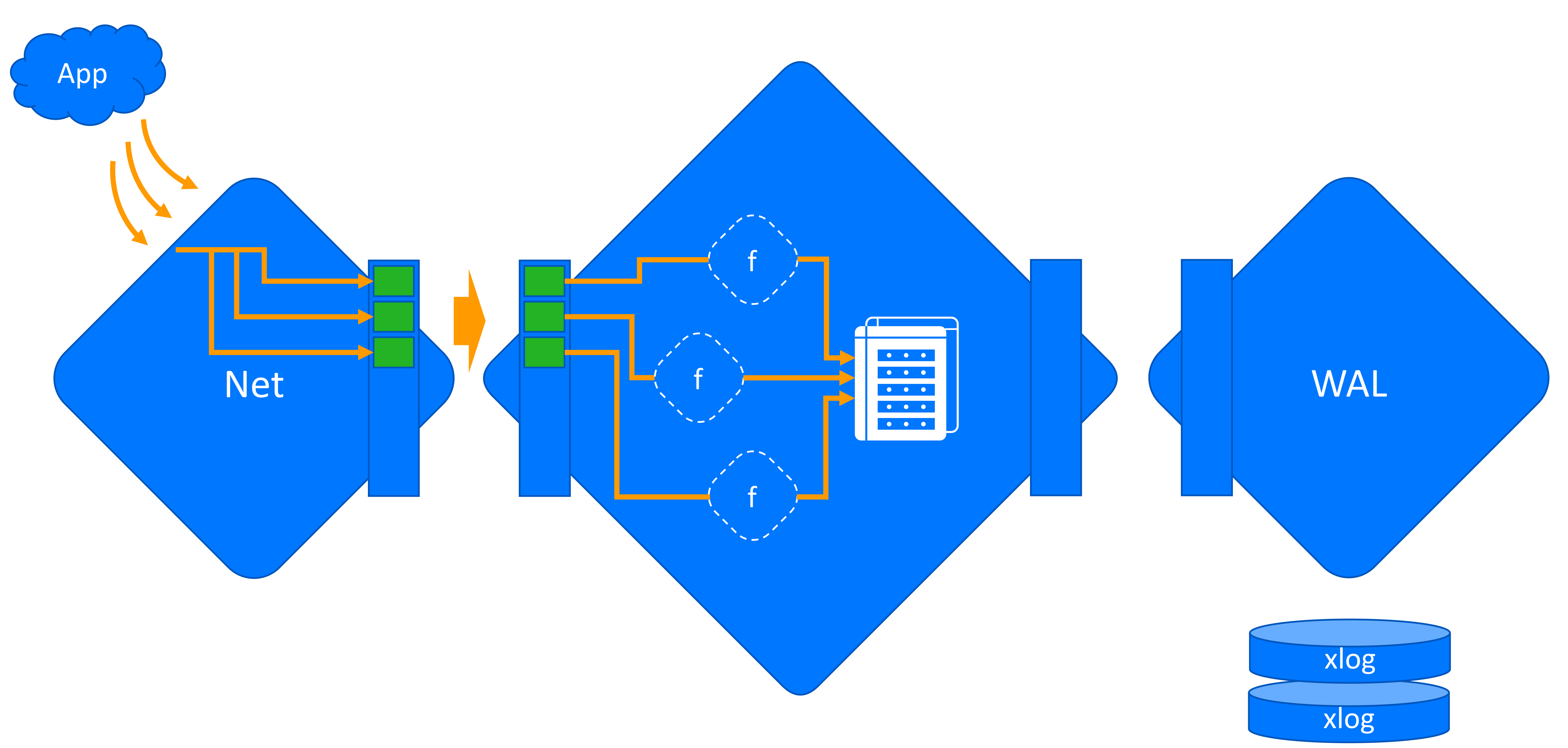

Net
f
f
TX
f
WAL
xlog
xlog


buffers
App
Net
f
f
TX
f
WAL
xlog
xlog

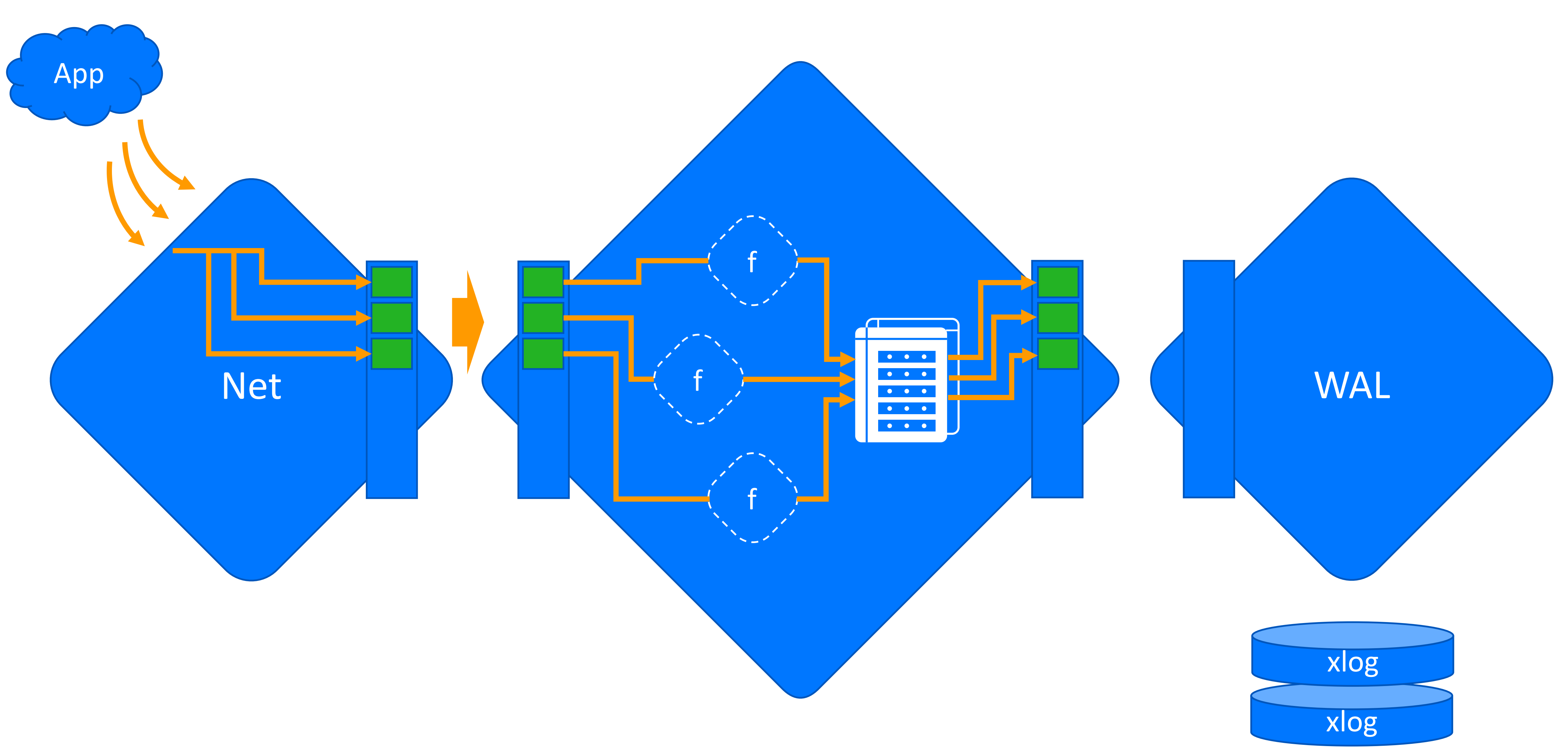

App
Net
f
f
TX
f
WAL
xlog
xlog

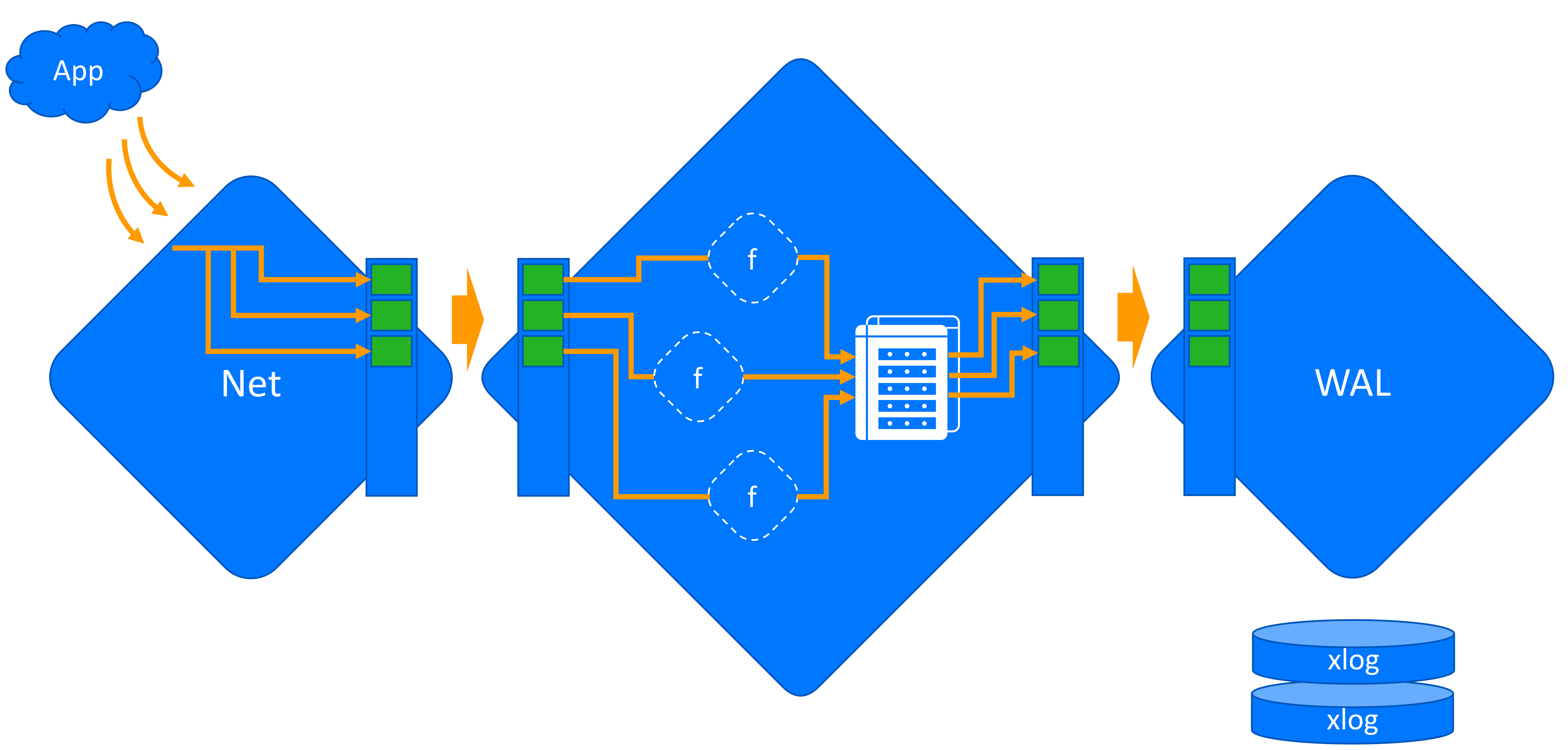


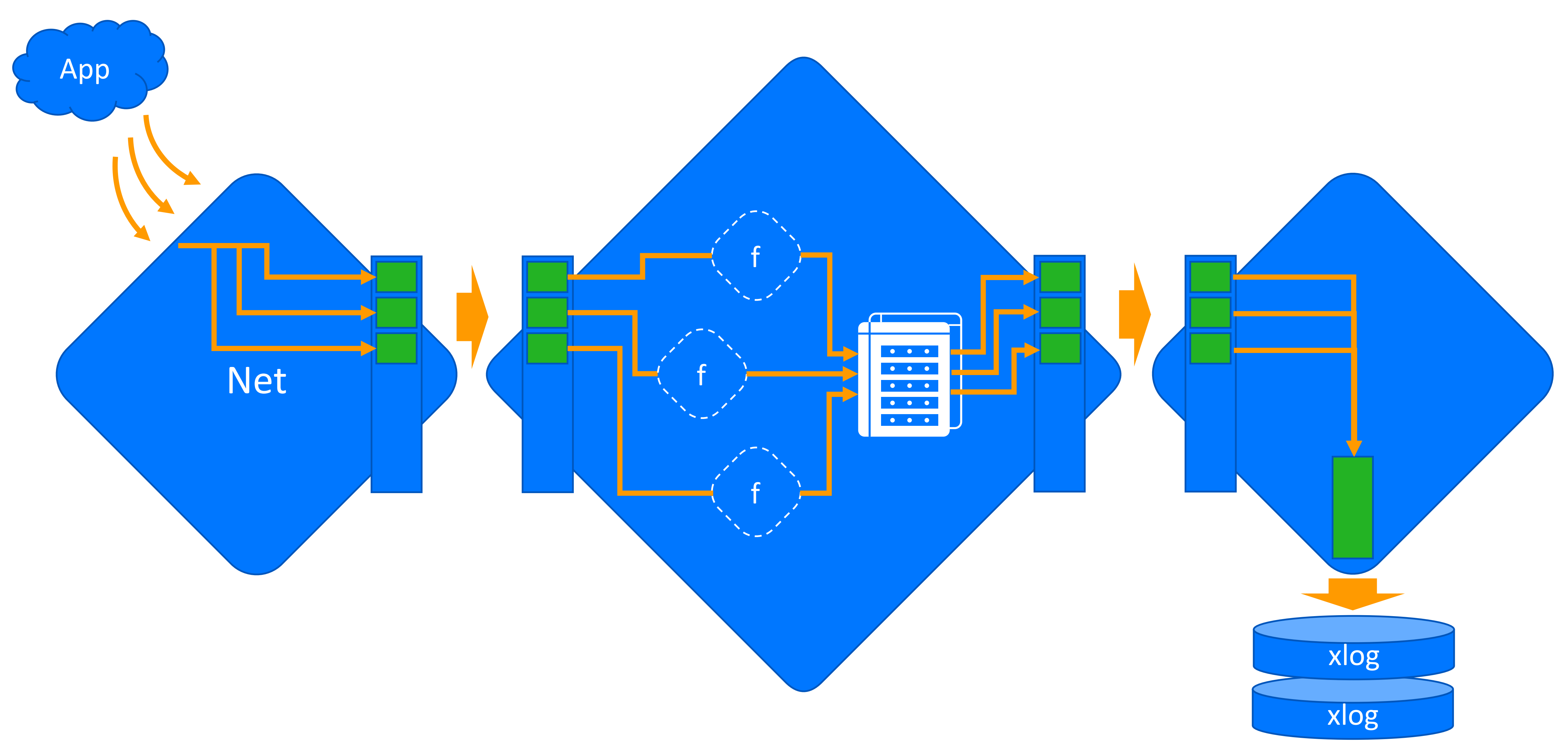
App
1 syscall
Net
f
f
TX
f
WAL
xlog
xlog

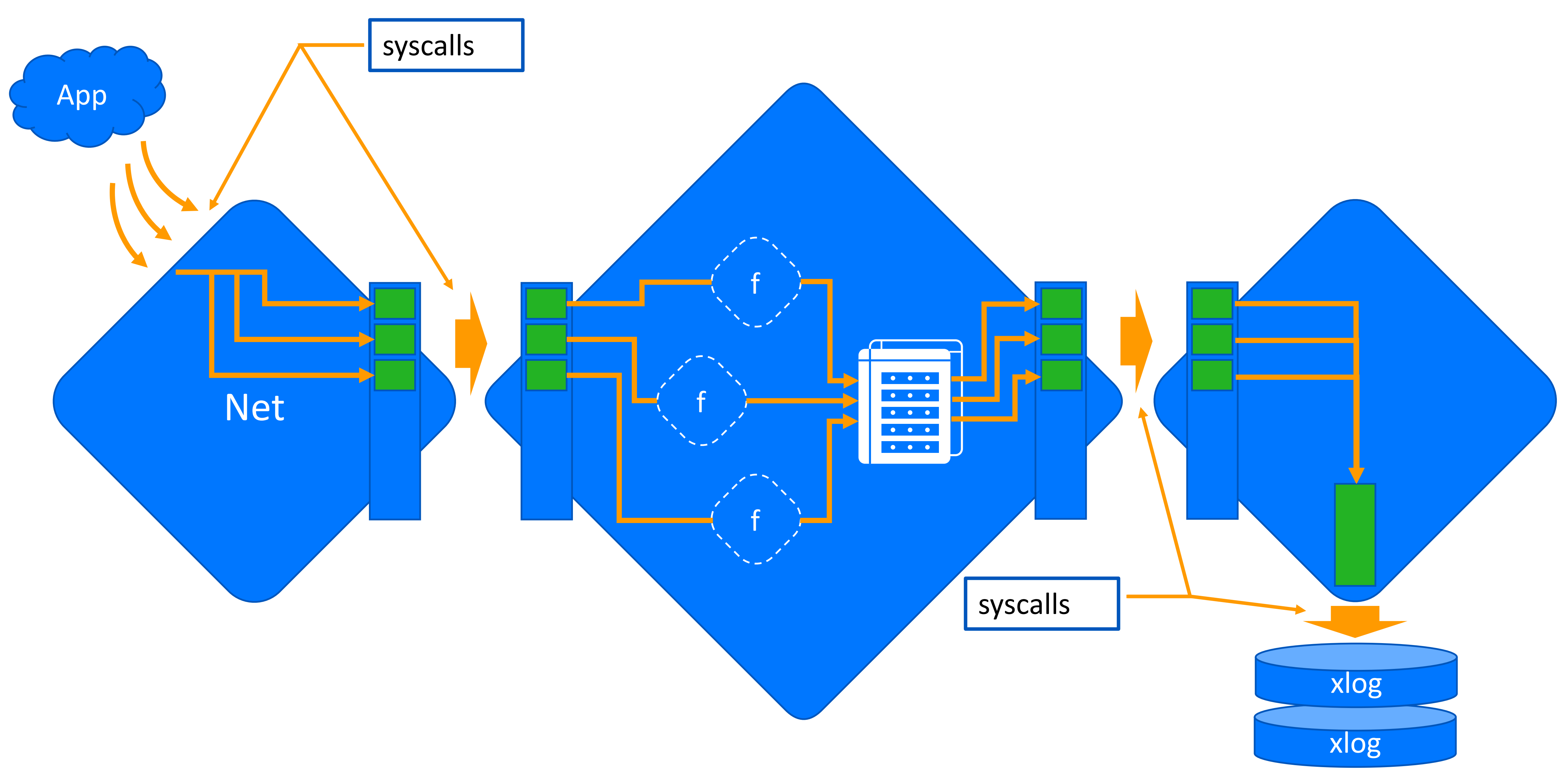


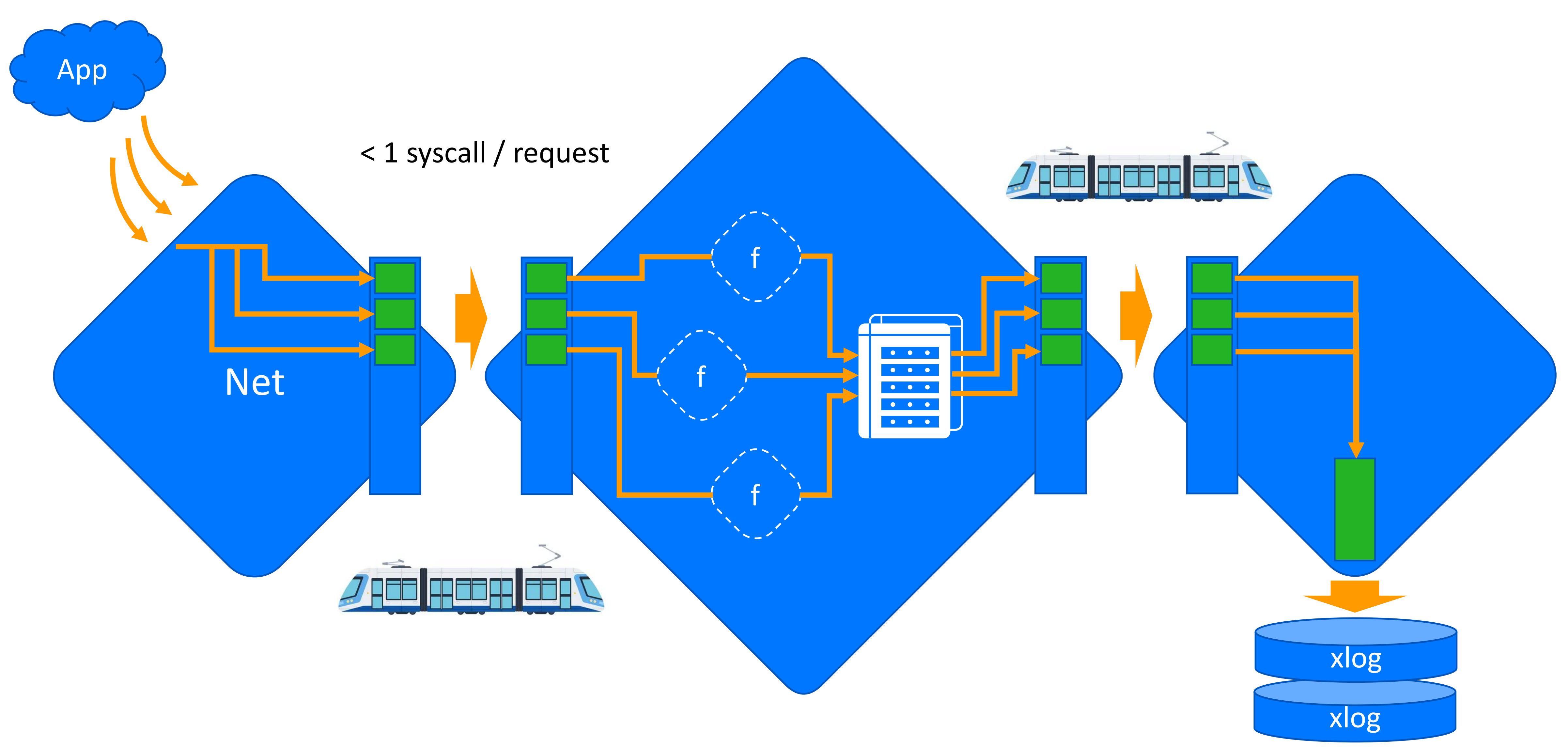
App
Net
f
f
f
WAL
xlog
xlog

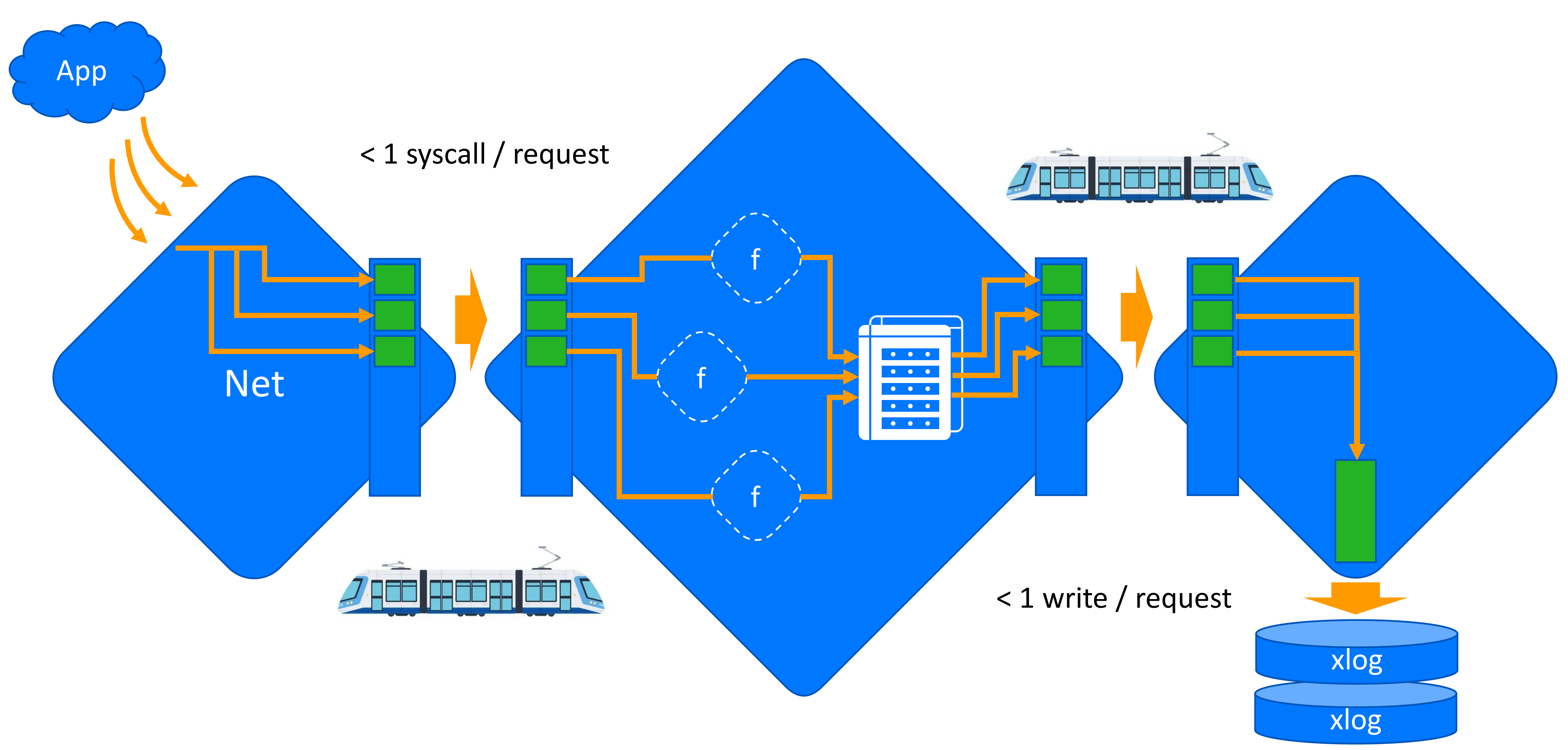

App
Net
f
f
f
WAL
xlog
xlog

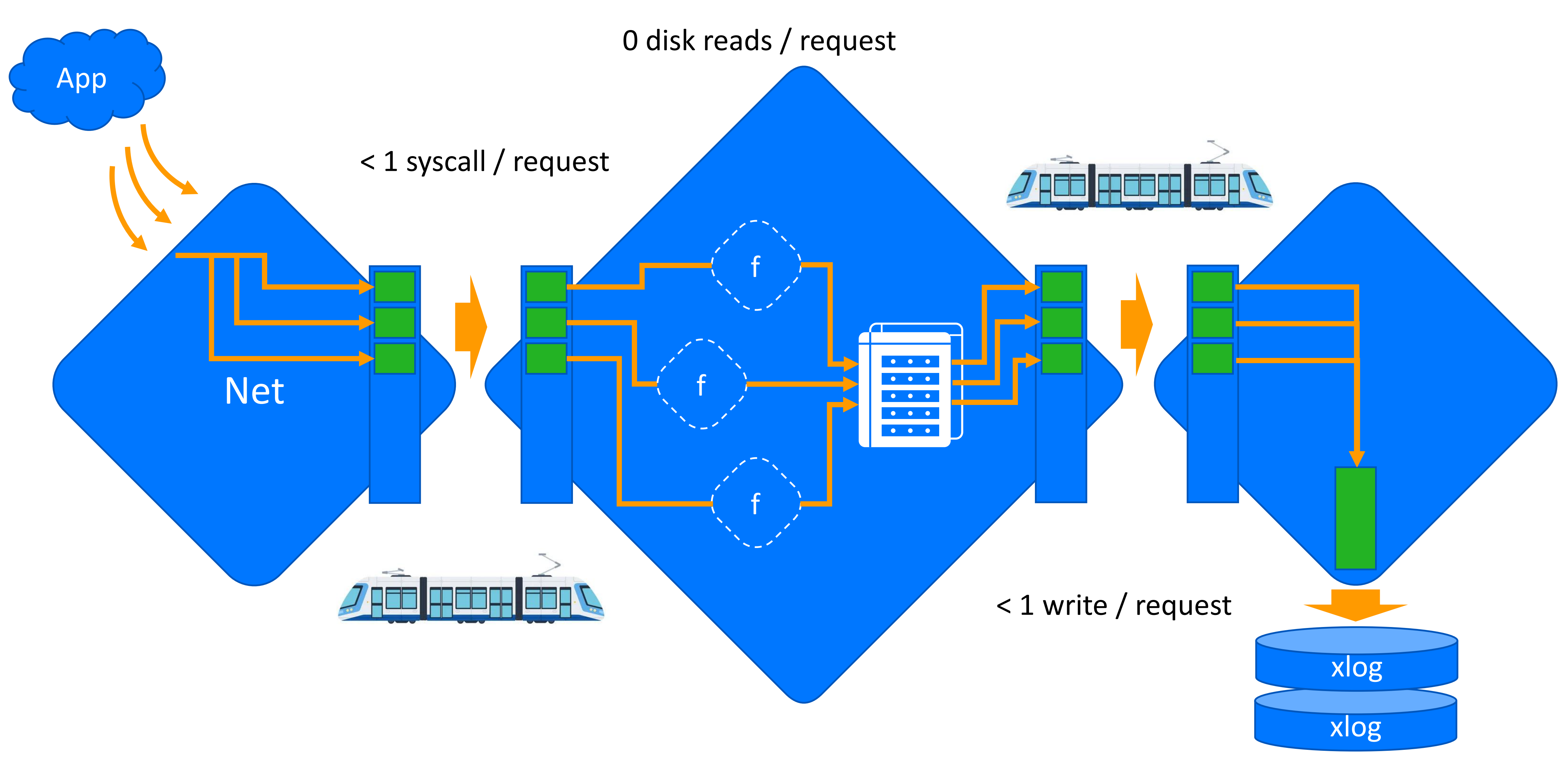

App
Net
f
f
f
WAL
xlog
xlog


App
Net
f
f
f
xlog
xlog


syscalls
App
Net
f
f
f
syscalls
xlog
xlog


App
< 1 syscall / request
Net
f
f
f
xlog
xlog


App
< 1 syscall / request
Net
f
f
f
< 1 write / request
xlog
xlog


App
0 disk reads / request
< 1 syscall / request
Net
f
f
f
< 1 write / request
xlog
xlog

Сохранность данных в памяти

Запись «ждёт»

Пишем в память

Пишем на диск

Ждём подтверждения

Сохранность данных в памяти

Чтение «не ждёт»

Читаем из памяти

Гарантированно быстро

Запись «ждёт»

Пишем в память

Пишем на диск

Ждём подтверждения





Восстановление после перезапуска

- Чтение снапшота
- Чтение xlog
- Построение индексов





Восстановление после перезапуска

- Чтение снапшота
- Чтение xlog
- Построение индексов

Долгий старт





03

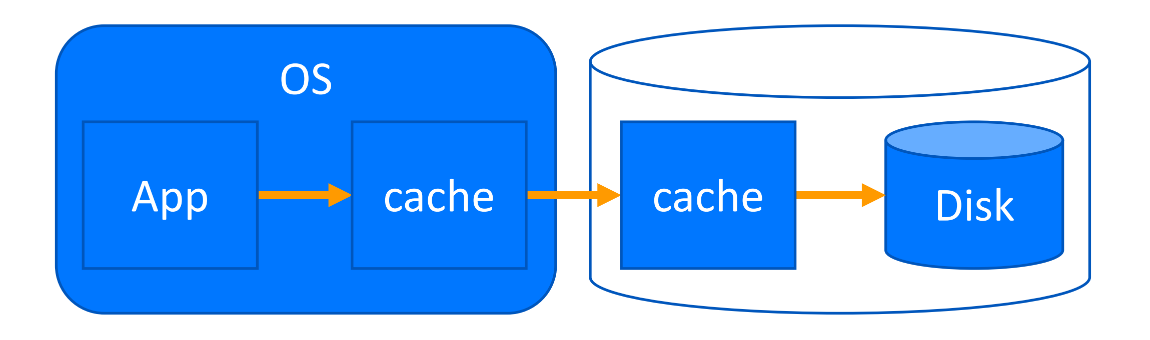
Обеспечение сохранности данных

Персистентность, диск и миф про fsync

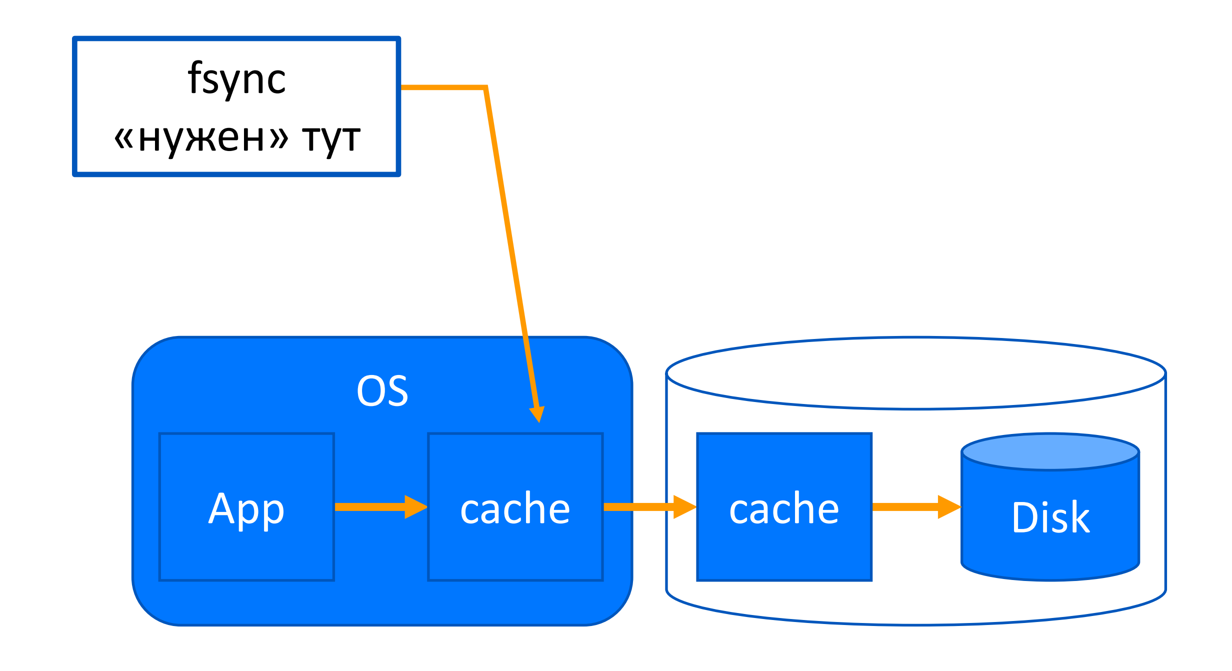
Для чего нужен fsync?



Для чего нужен fsync?



Для чего нужен fsync?





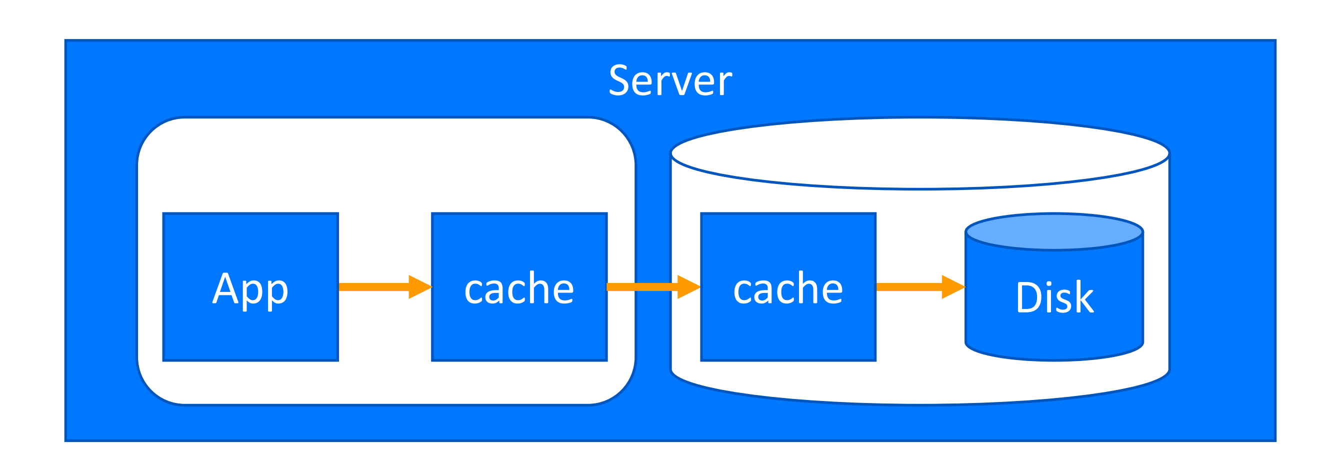


Для чего нужен fsync?

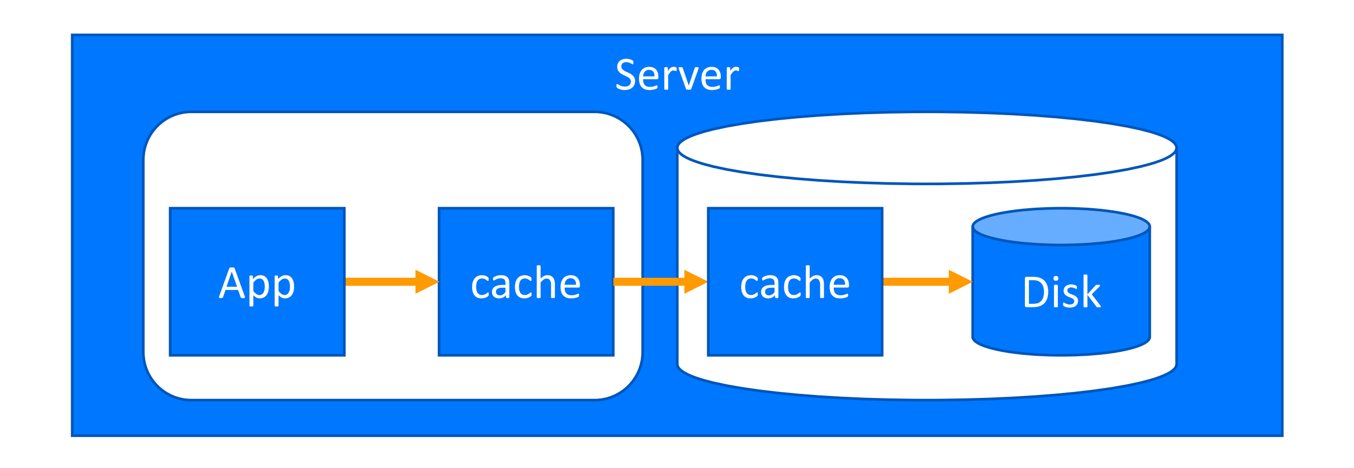


Для чего нужен fsync?





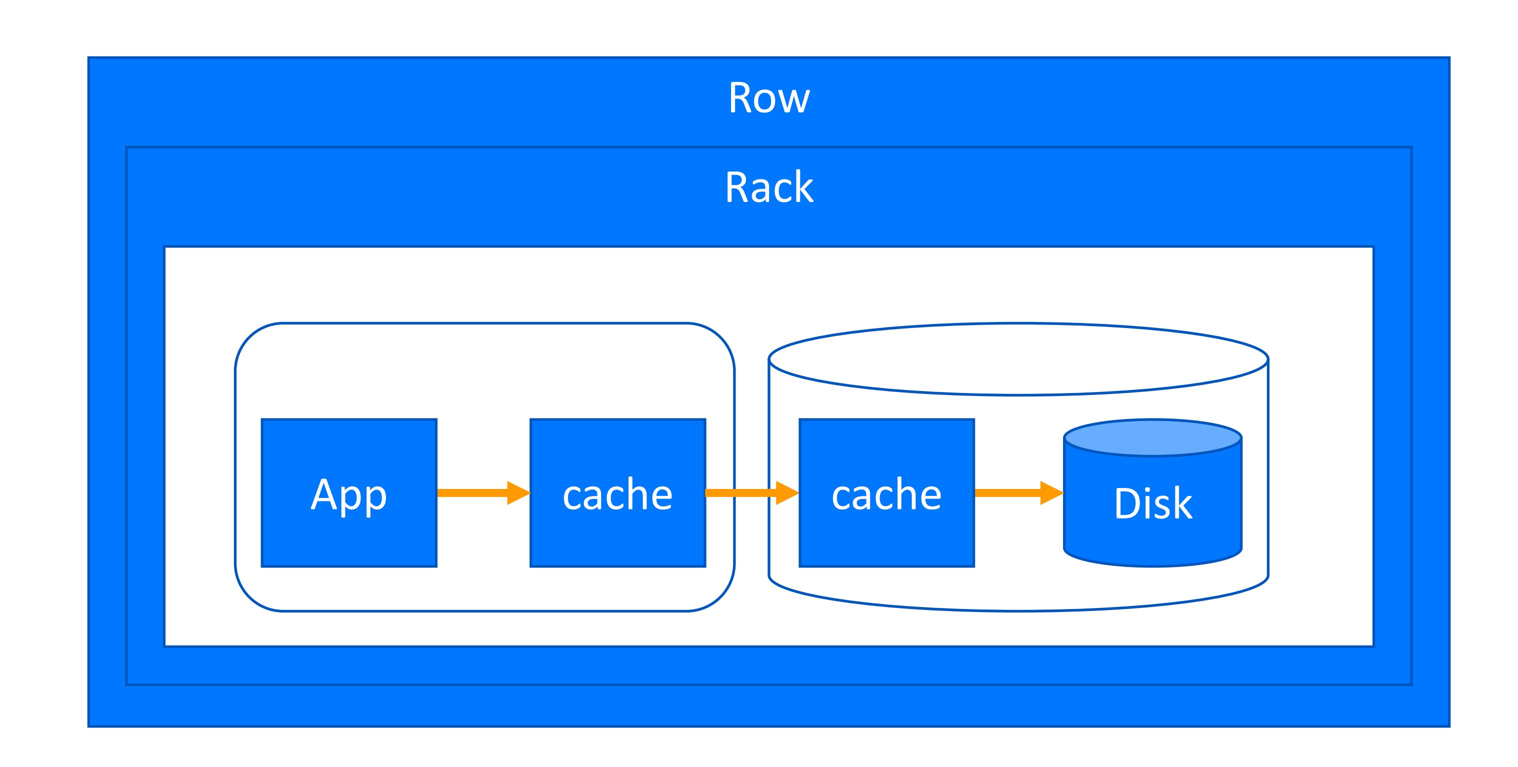
Для чего нужен fsync?







Для чего нужен fsync?

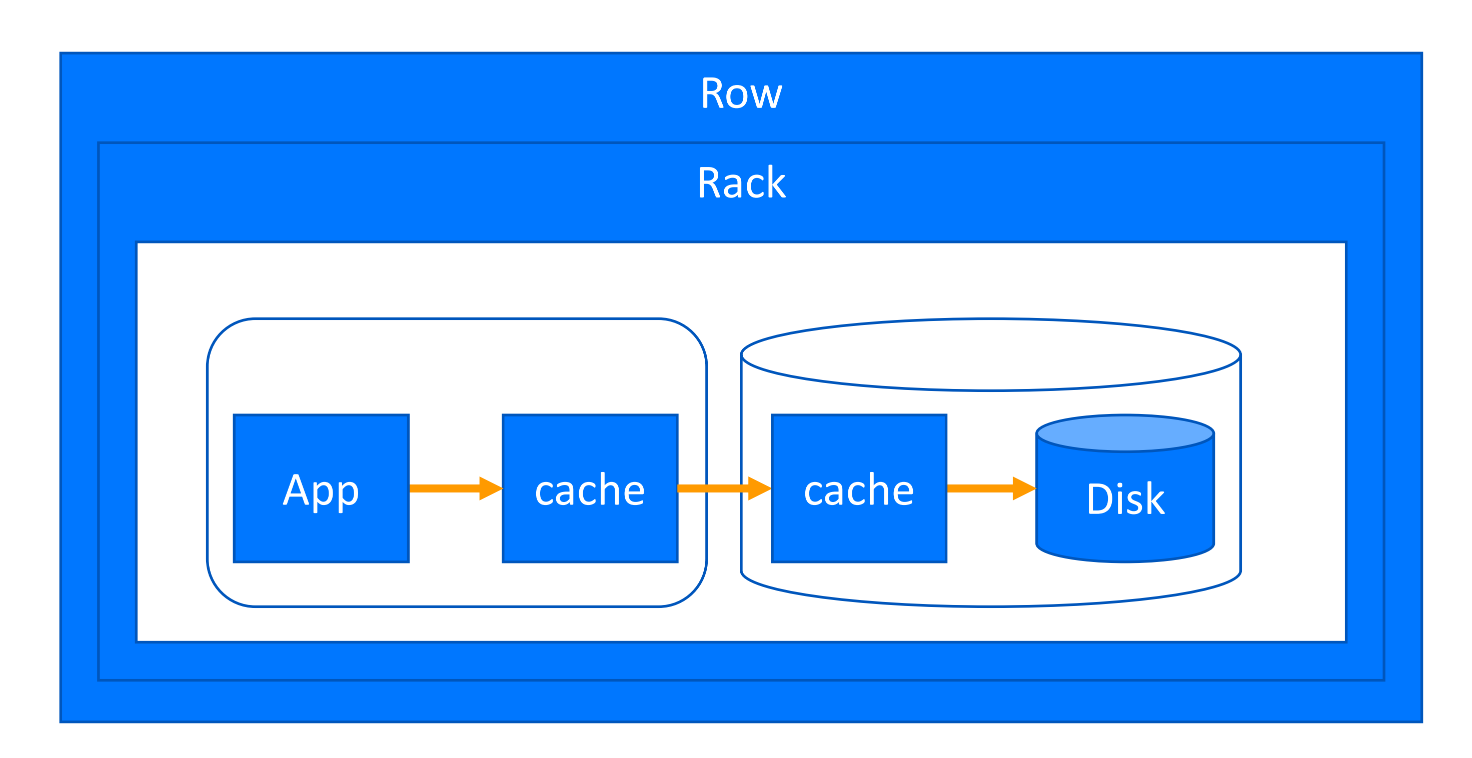






Для чего нужен fsync?



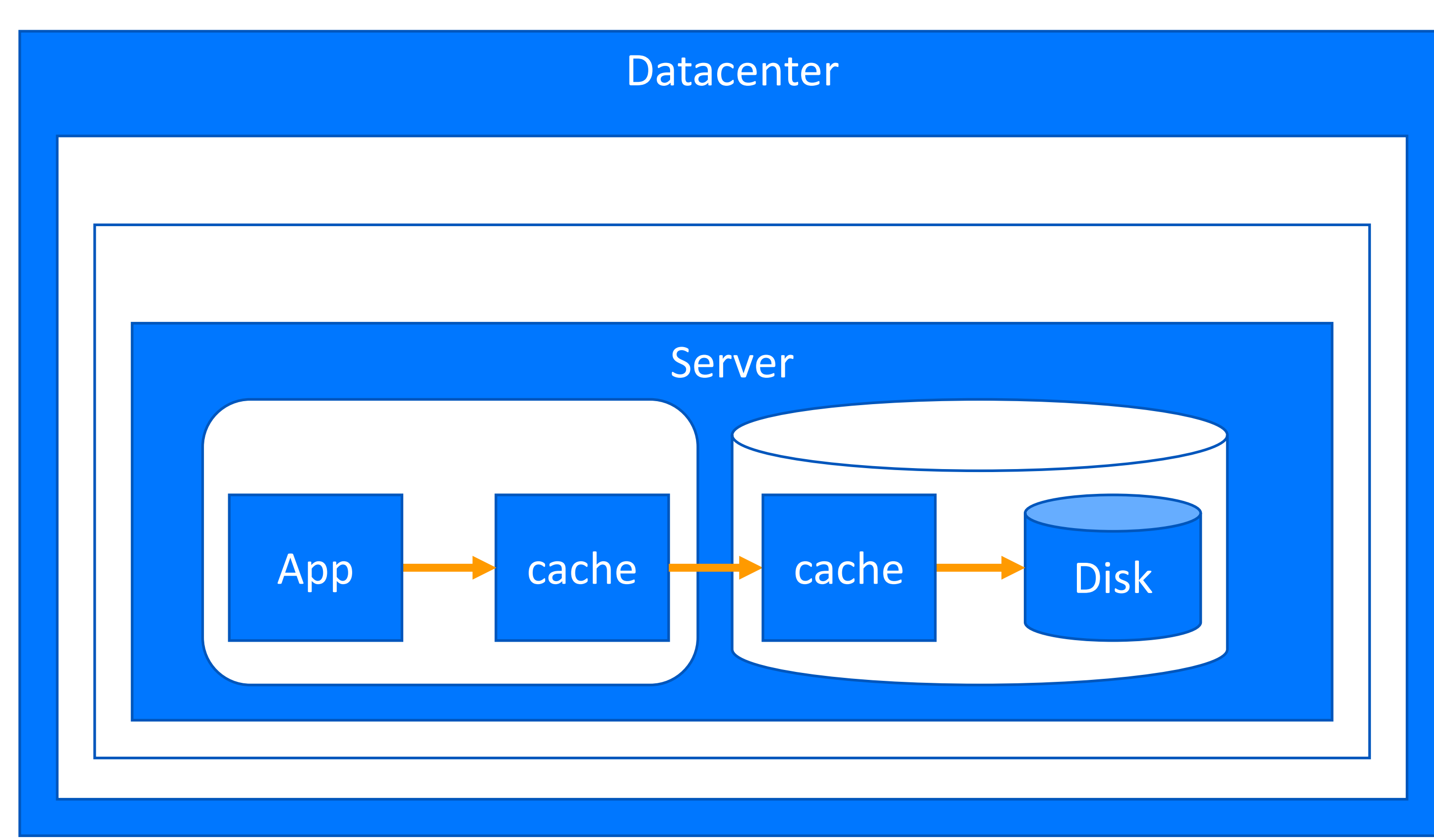






Для чего нужен fsync?









Датацентры горят









fsync защищает слишком слабо

Не защищает от отказа диска

Не защищает от отказа сервера

Не защищает от отказа датацентра





fsync защищает слишком слабо

Не защищает от отказа диска

Не защищает от отказа сервера

Не защищает от отказа датацентра

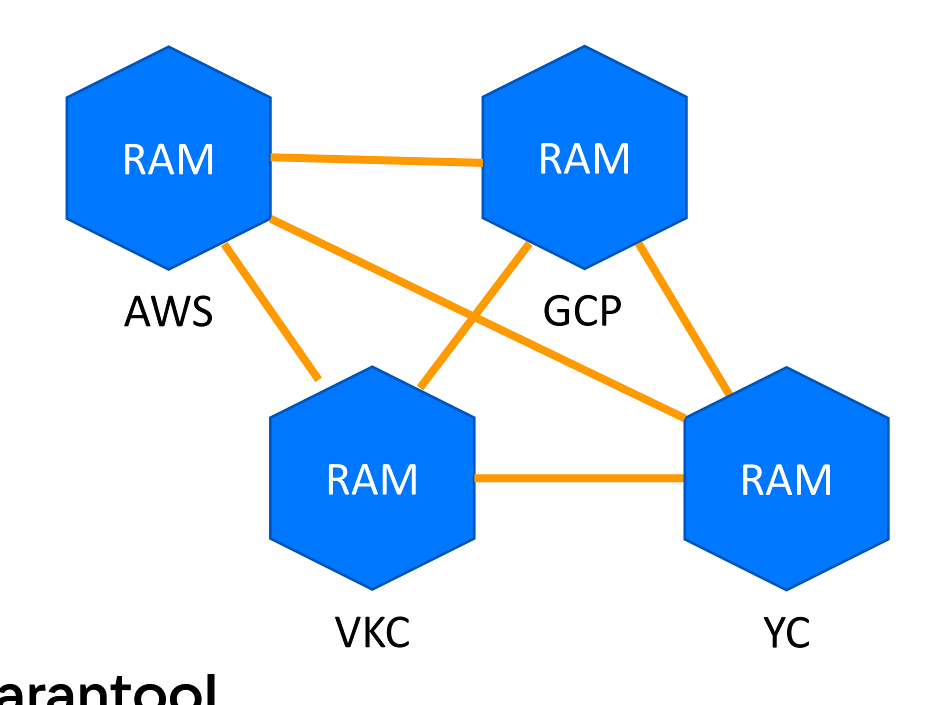
Поэтому нужна репликация

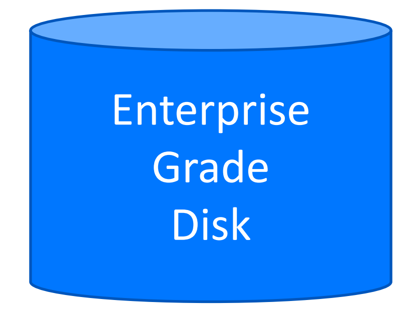




Память против дисков

Распределённая и задублированная система, не сохраняющая на диск, будет надёжнее одного диска*









Вероятность одновременного отказа

$$Pn = P^n$$

- 1 — 50%





Вероятность одновременного отказа

$$Pn = P^n$$

- 1 — 50%
- 2 — 25%
- 3 — 12.5%
- 4 — 6.25%

Вероятность одновременного отказа

$$Pn = P^n$$

- 1 — 50%
- 2 — 25%
- 3 — 12.5%
- 4 — 6.25%
- 10 — 0.1%





Сохранность в памяти + репликация

Читаем из памяти

Пишем в память

Пишем на диск

Ждём подтверждения

Реплицируем

(Ждём подтверждения)





Сохранность в памяти + репликация

Потерять…
…все данные

RPO = «начало»

Потерять…
… «последние» данные

RPO = «последние»





Сохранность в памяти + репликация

Потерять…
…все данные

RPO = «начало»

Потерять…
… «последние» данные

RPO = «последние»







Сохранность в памяти + репликация

Потерять…
…все данные

RPO = «начало»

Потерять…
… «последние» данные

RPO = лаг репликации







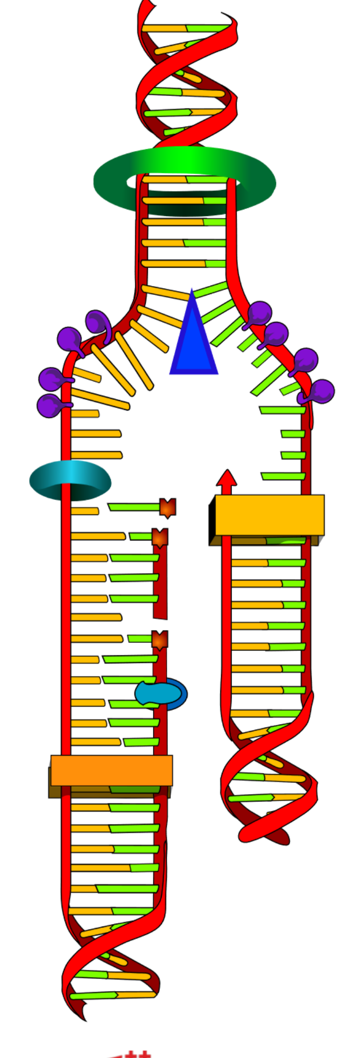
04

Репликация

Какие виды репликации бывают и почему так много?

Что такое репликация?

Механизм создания точной копии



Зачем нужна репликация?

Резервирование:

- Одно-нодовые системы ненадежны (диски, питание)
- Одного ДЦ недостаточно (пожар)



Зачем нужна репликация?

Резервирование:

- Одно-нодовые системы ненадежны (диски, питание)
- Одного ДЦ недостаточно (пожар)

Доступность:

- Время восстановления после аварии (RTO)





Зачем нужна репликация?

Резервирование:

- Одно-нодовые системы ненадежны (диски, питание)
- Одного ДЦ недостаточно (пожар)

Доступность:

- Время восстановления после аварии (RTO)

Производительность:

- Железо узкое место (1 thread, RAM)





Какая бывает репликация

Синхронная — надёжность

Асинхронная — скорость



Синхронная и асинхронная

Синхронная	Асинхронная
подтверждается после фиксации на реплике	подтверждается после записи на одну ноду
надежная (durable)	можно потерять часть
медленная	«быстрая»
выше шанс отказа	отказ реплики не важен
RPO = 0	**RPO > 0 (≈ Lag)**

* RPO — Recovery Point Objective



Какая бывает репликация

Однонаправленная — консистентность

Двунаправленная — доступность

Master-master vs Master-replica

MM

сложно использовать

RTO = 0

MR

просто использовать

RTO > 0 (≈ Failover time)

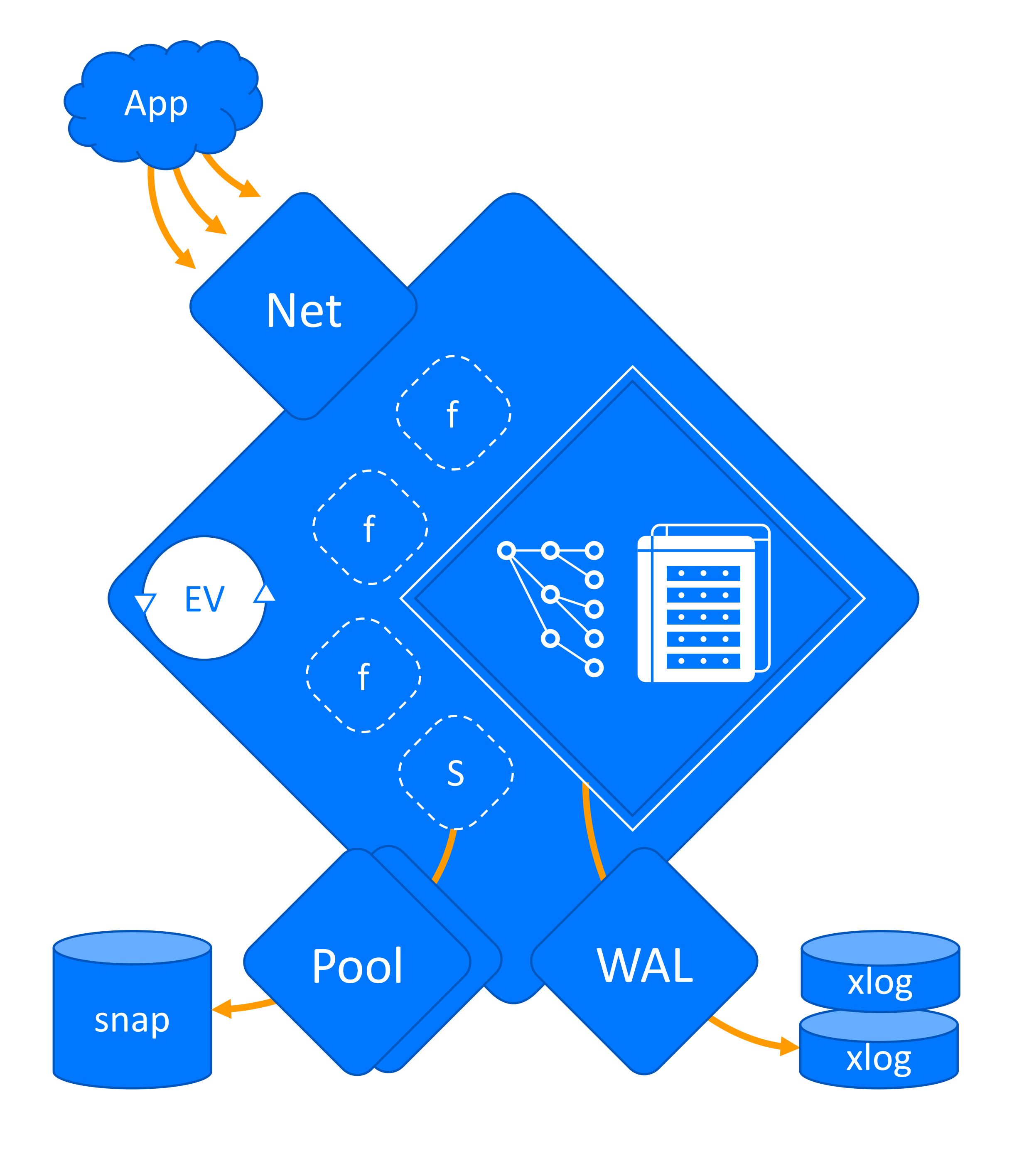
* RTO — Recovery Time Objective





Устройство репликации в Tarantool





- Данные в памяти
- Монопольный доступ
- Событийный цикл
- Кооперативная многозадачность
- Отдельные потоки для сети
- Изменения во Write-Ahead-Log
- Консистентный снапшот
- Отдельные потоки для диска
- **WAL реплицируется**

App

Net

f

f

EV

f

S

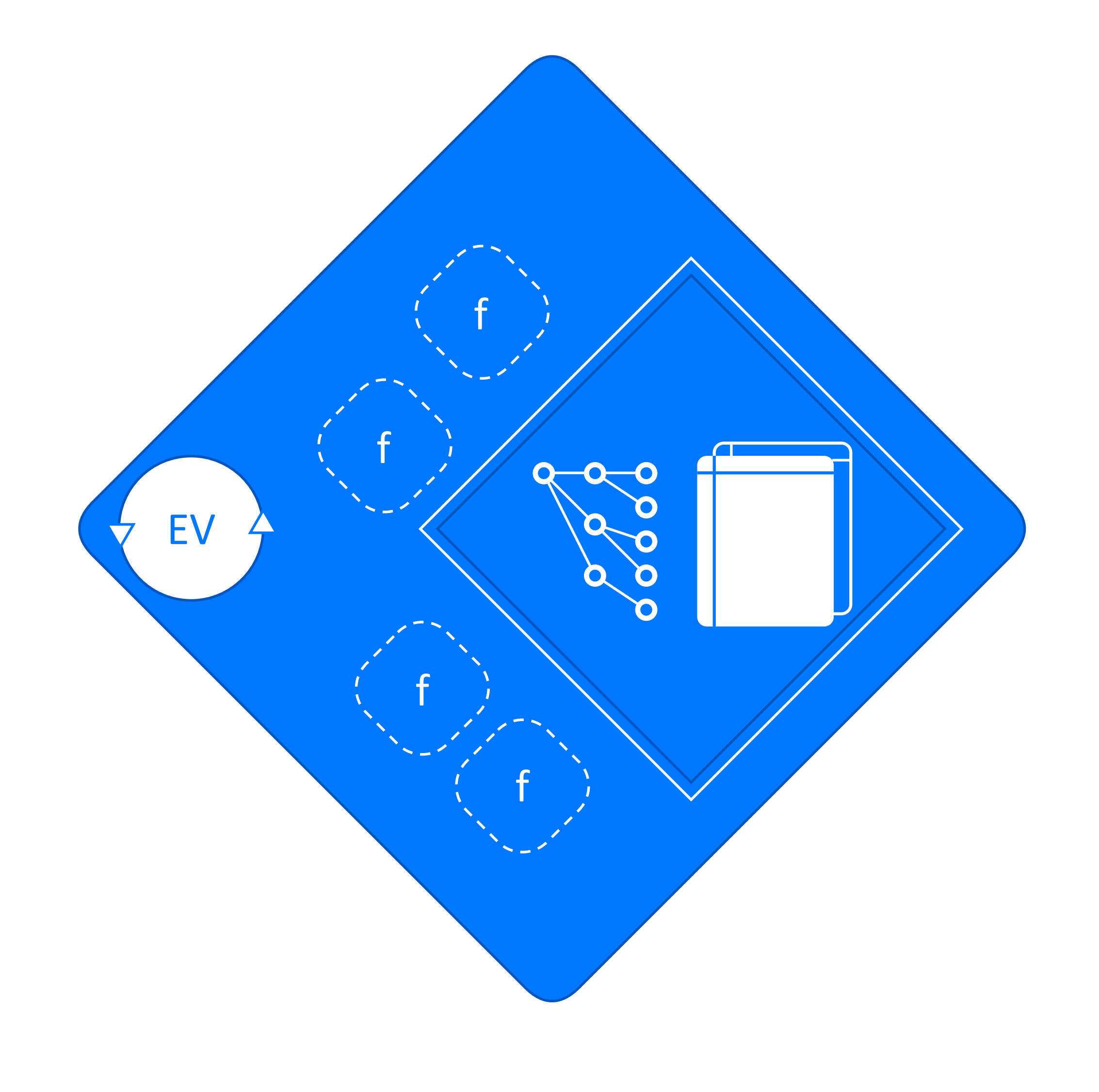
Pool

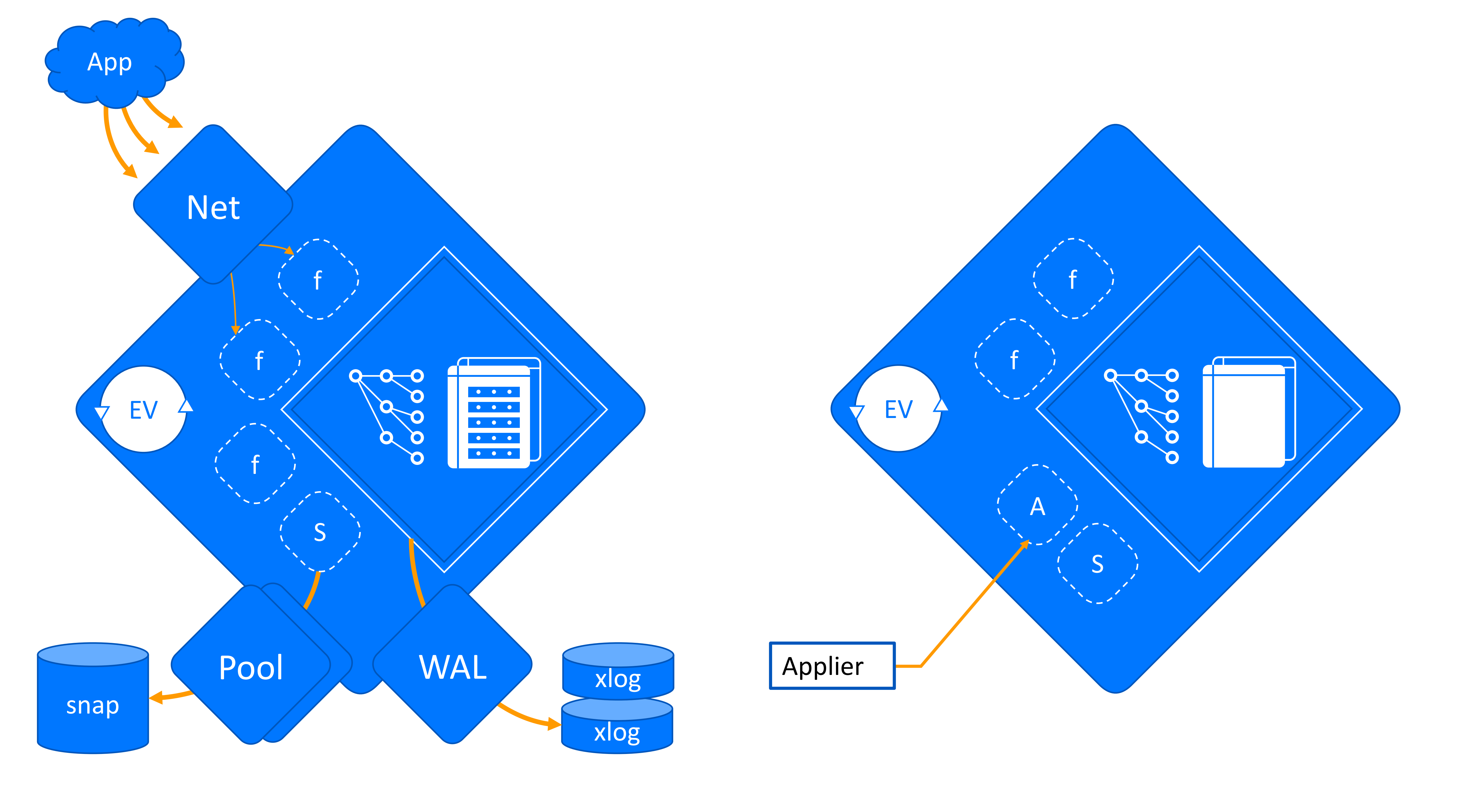
WAL

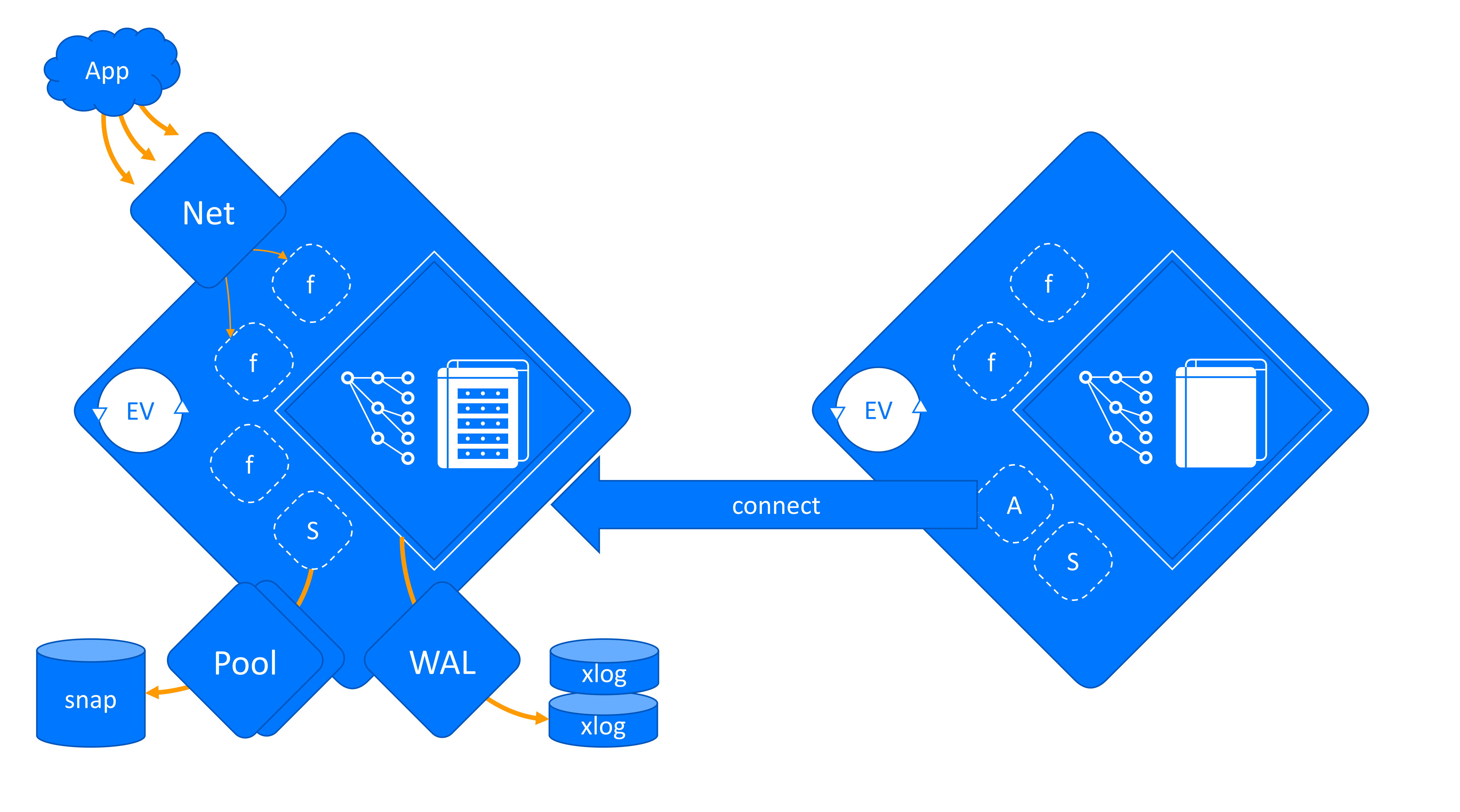
snap

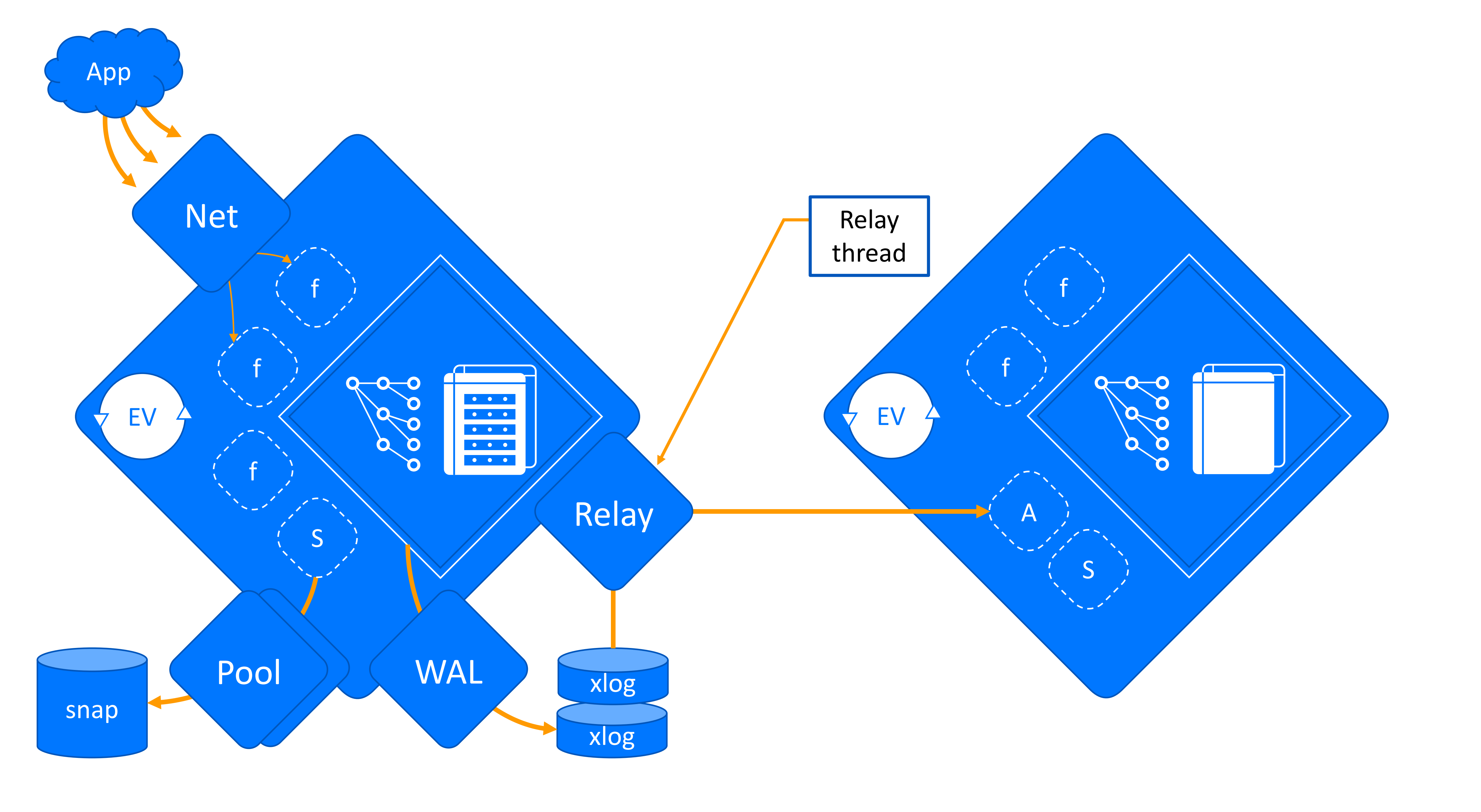
xlog

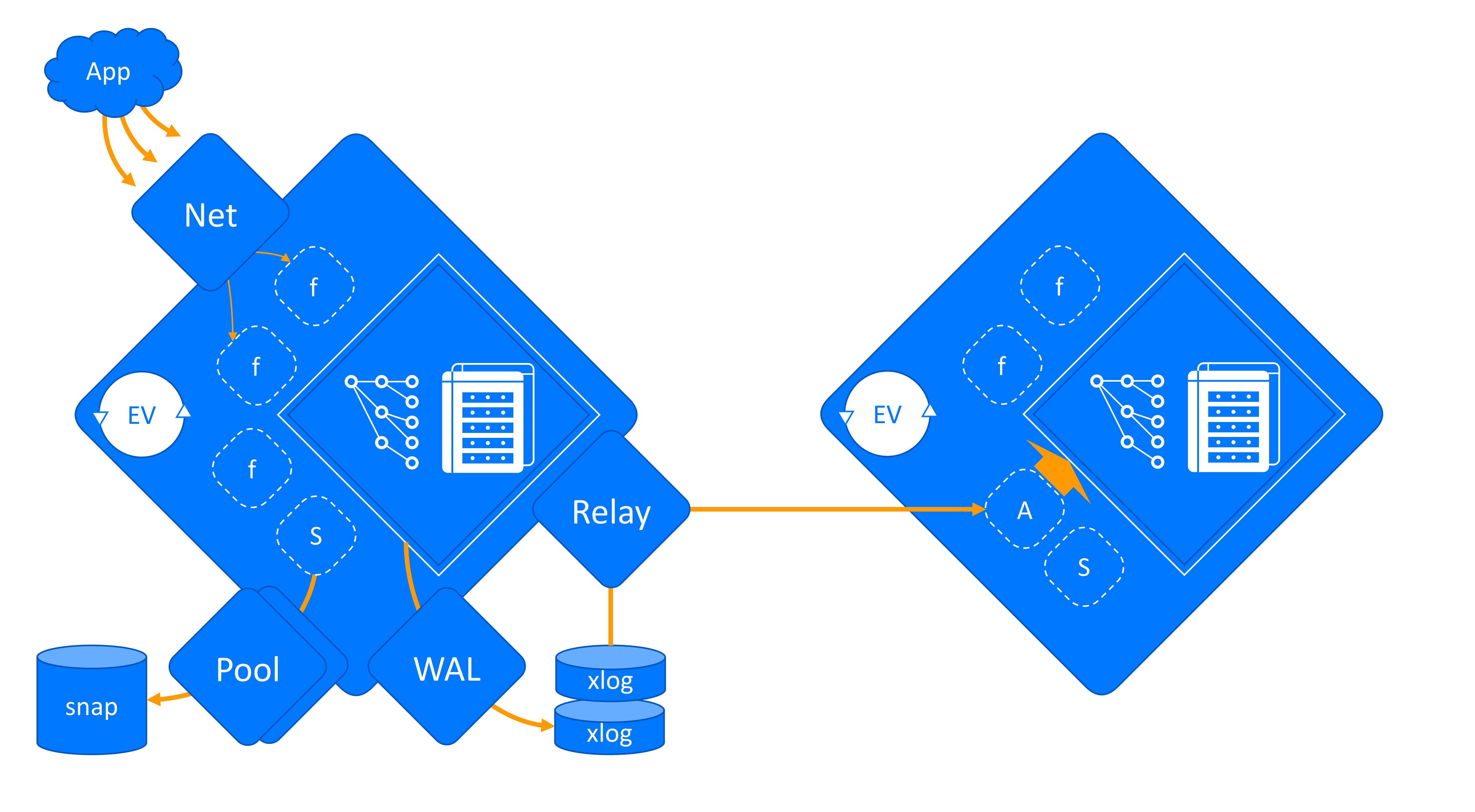
xlog

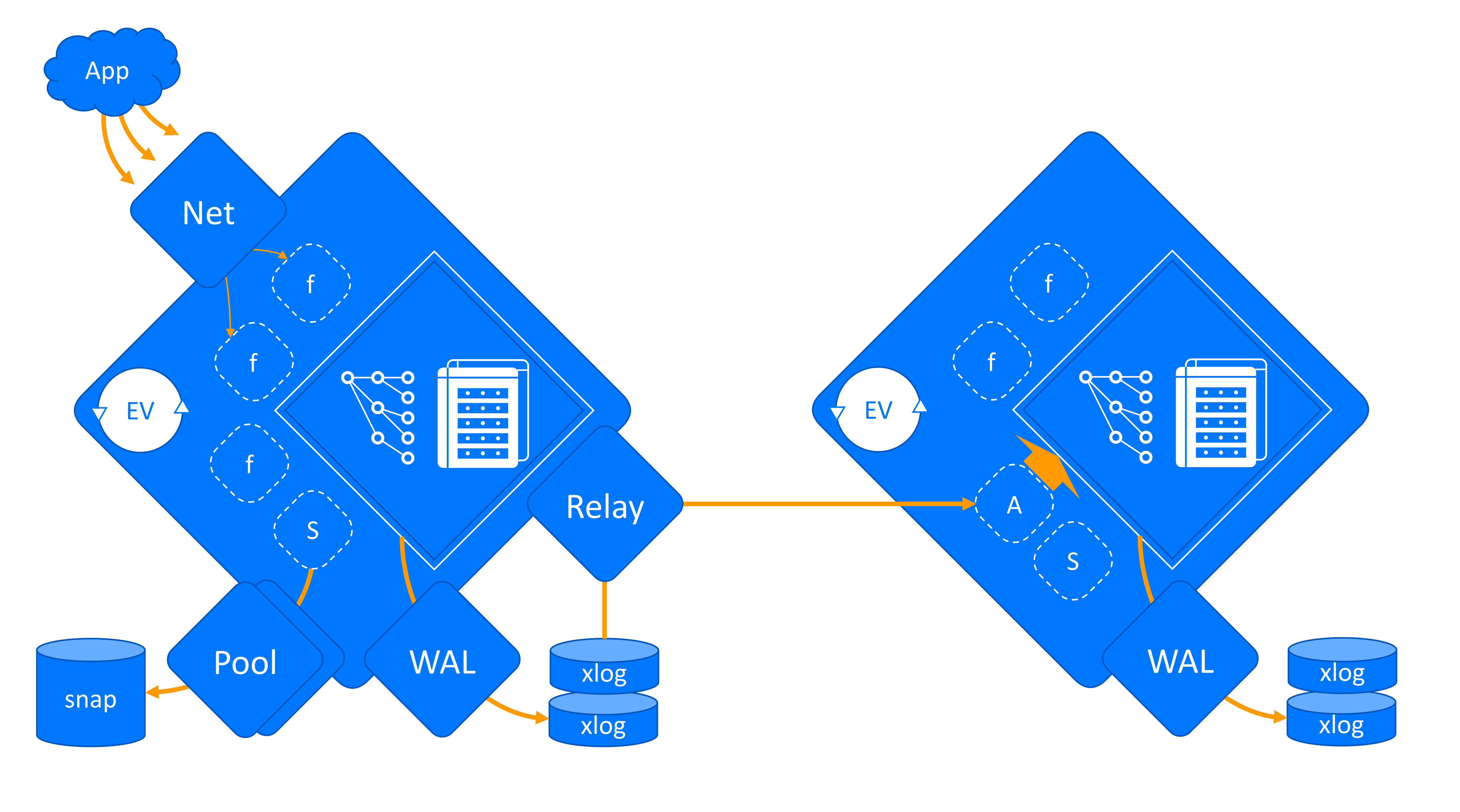


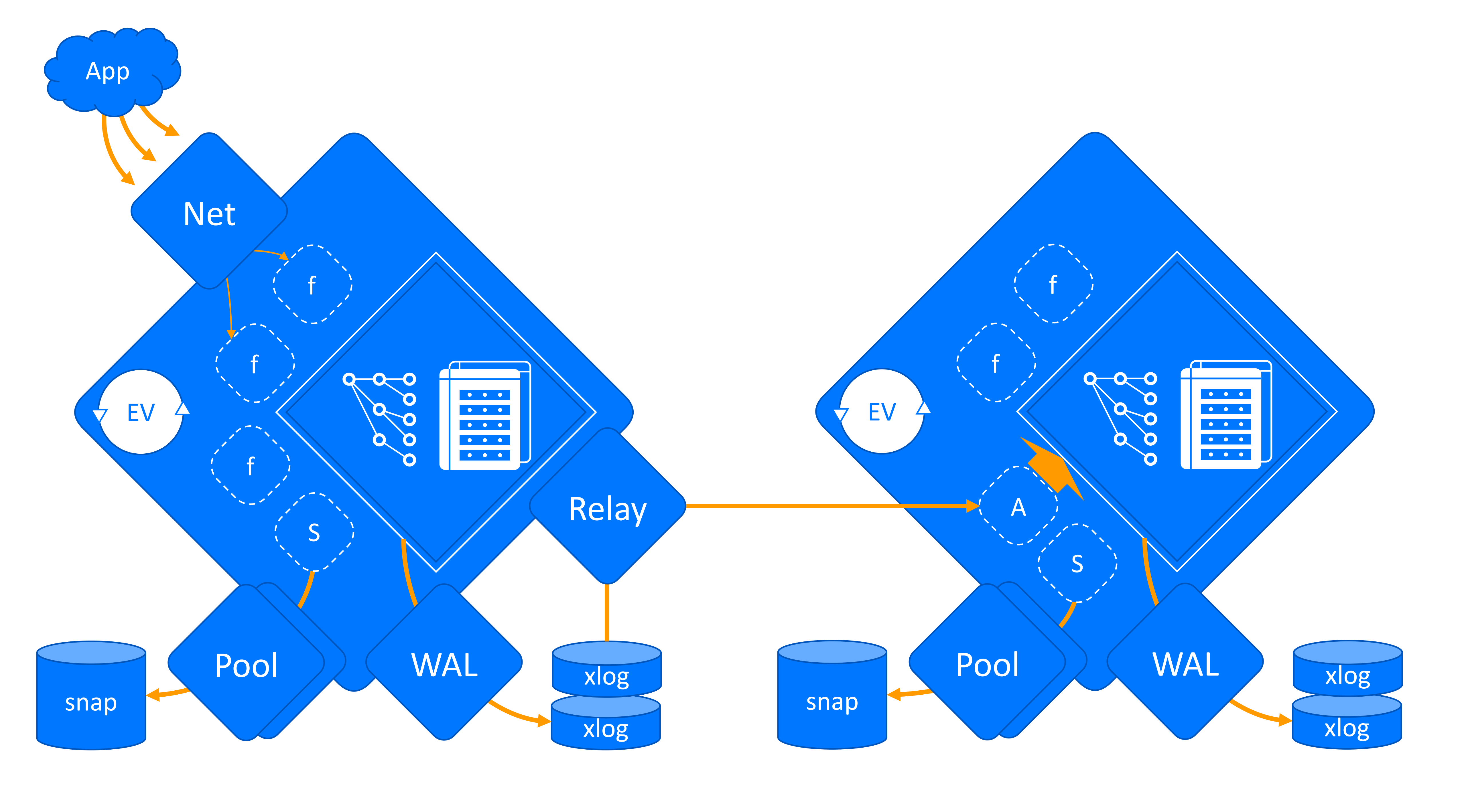

App
Net
f
f
f
EV
S
Pool
WAL
snap
xlog
xlog
f
f
EV
A
S
Applier

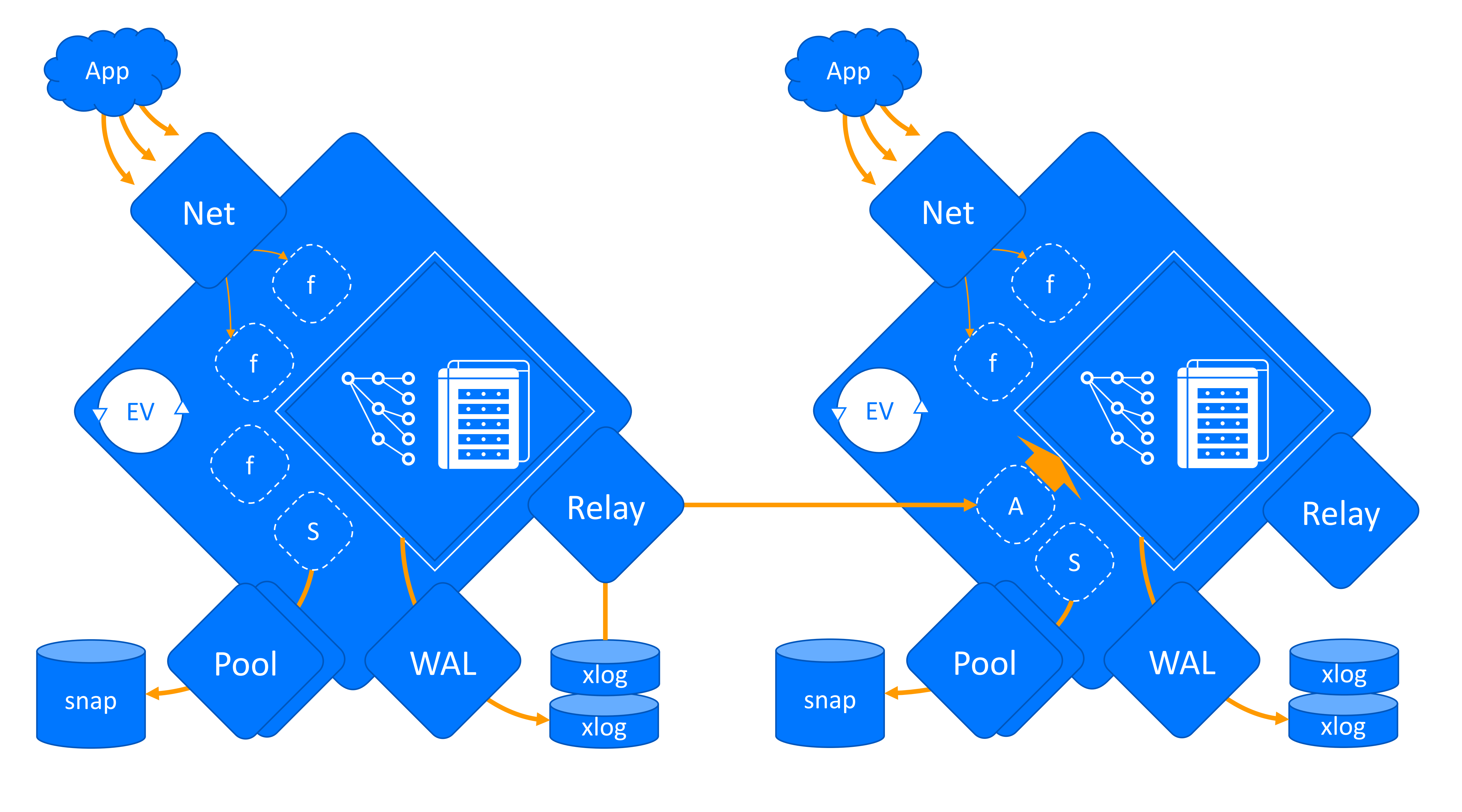

App
Net
f
f
f
S
EV
Pool
WAL
snap
xlog
xlog
connect
f
f
EV
A
S

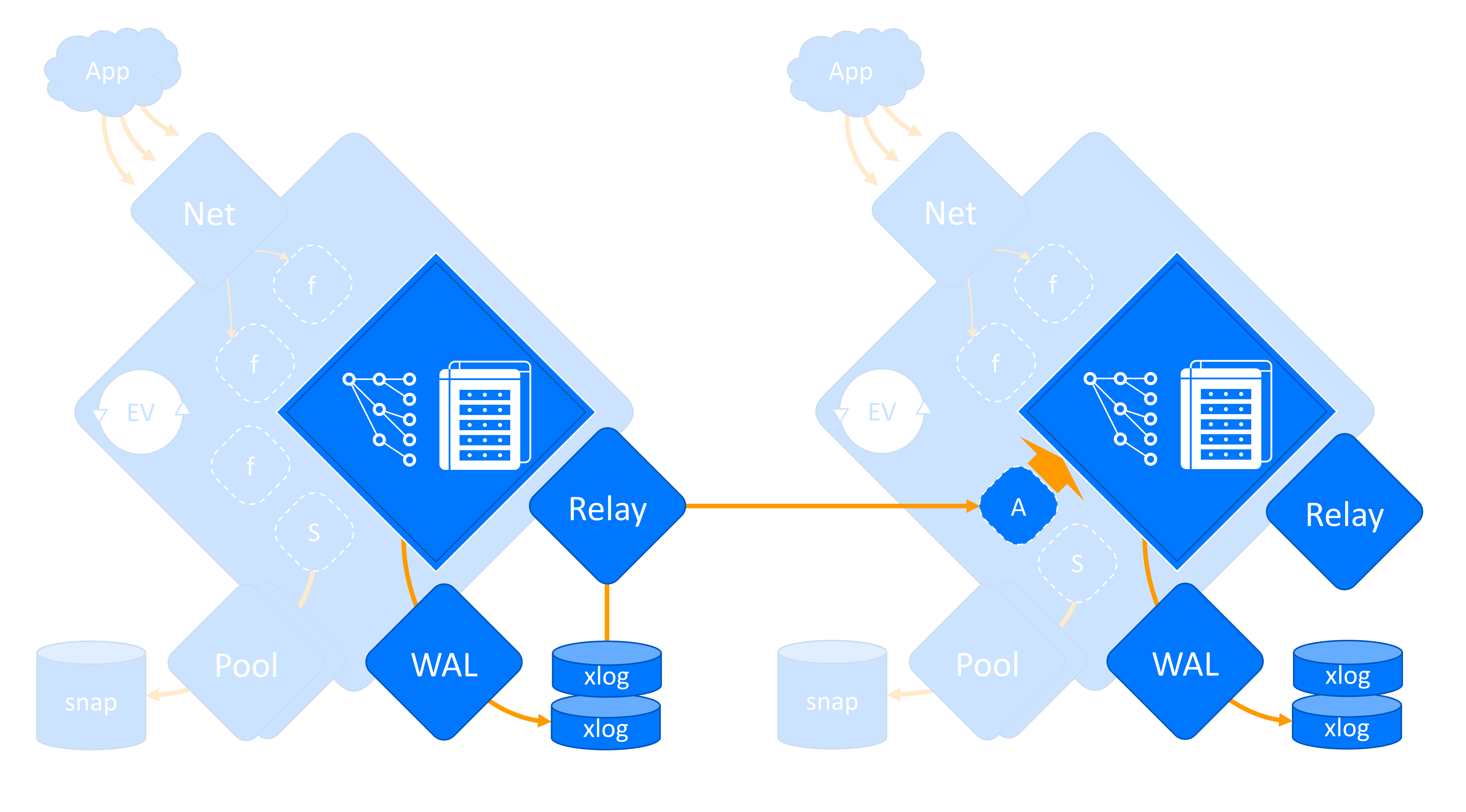

App
Net
f
f
f
S
EV
Relay thread
Relay
Pool
WAL
snap
xlog
xlog
f
f
EV
A
S


App
Net
f
f
f
EV
S
Relay
Pool
WAL
snap
xlog
xlog
f
f
EV
A
S


App
Net
f
f
f
S
EV
Relay
Pool
WAL
snap
xlog
xlog
f
f
EV
A
S
WAL
xlog
xlog


App
Net
f
f
f
S
EV
Relay
Pool
WAL
snap
xlog
xlog
f
f
EV
A
S
Pool
WAL
snap
xlog
xlog


App
Net
f
f
f
EV
s
Relay
Pool
WAL
snap
xlog
xlog
App
Net
f
f
EV
A
S
Relay
Pool
WAL
snap
xlog
xlog

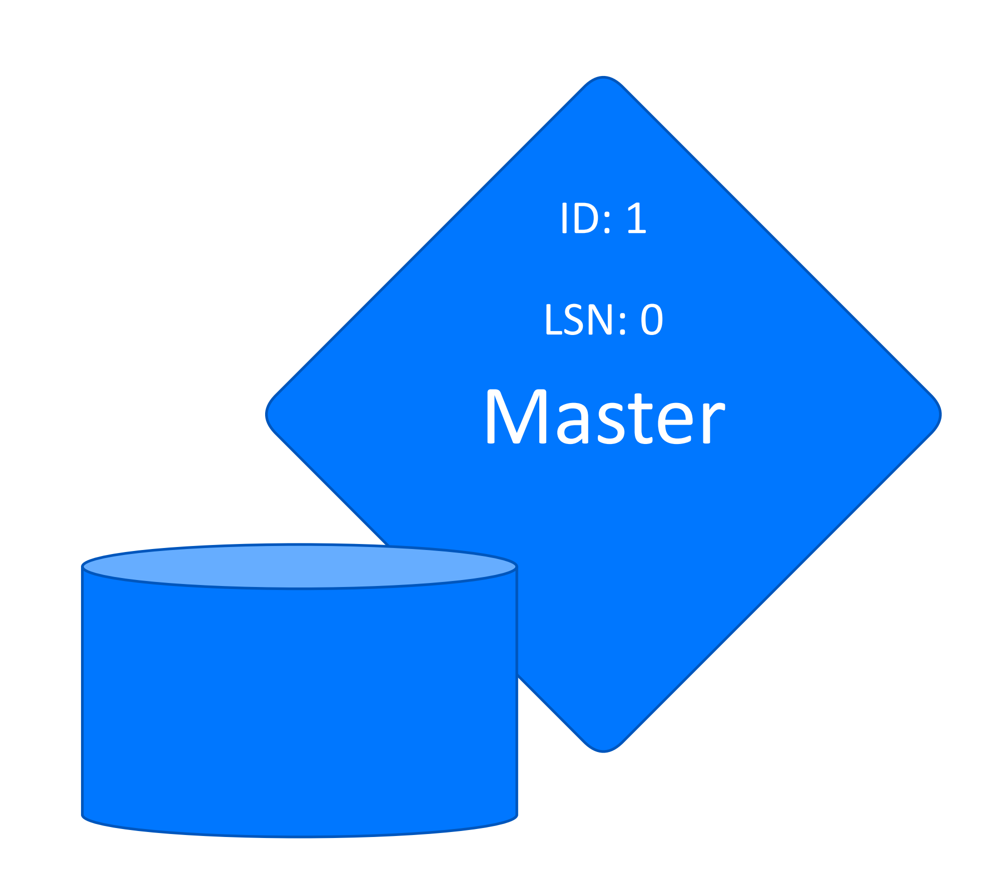

App
Net
f
f
EV
f
S
Relay
Pool
WAL
snap
xlog
xlog
App
Net
f
f
EV
A
S
Relay
Pool
WAL
snap
xlog
xlog

Log Sequence Number (LSN)

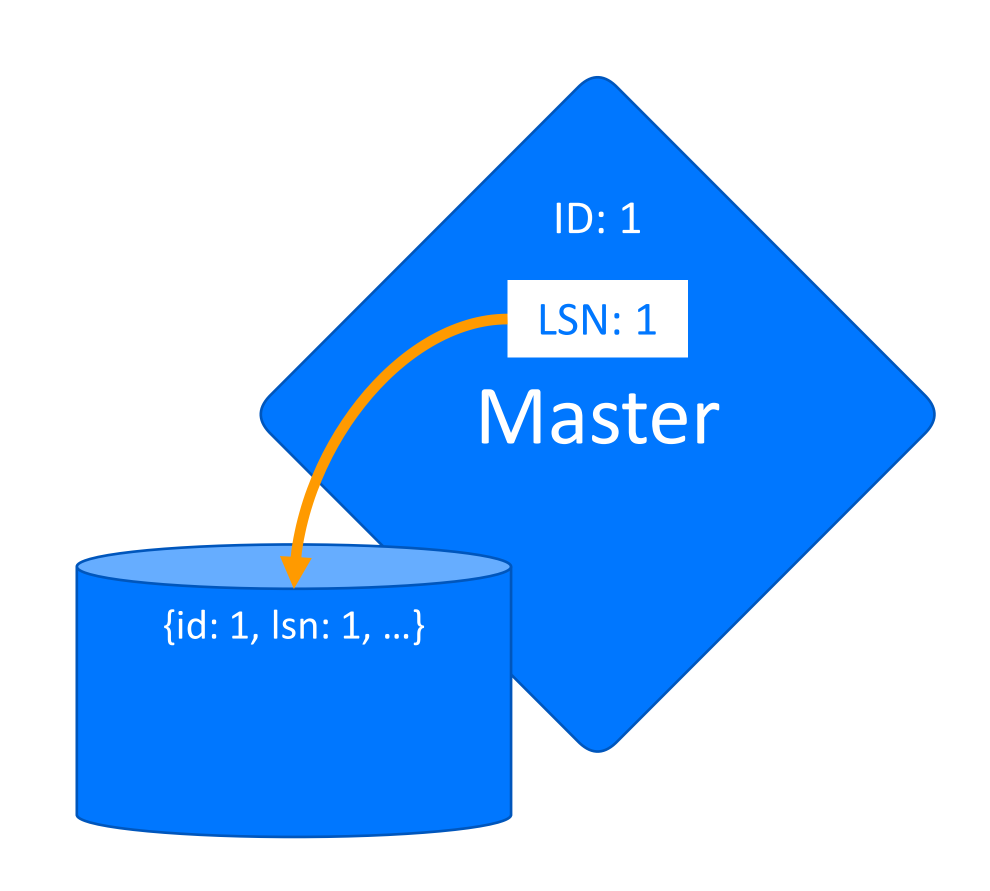
- Монотонная последовательность
- Определяет взаимный порядок транзакций
- Вместе с узлом — идентификатор транзакции



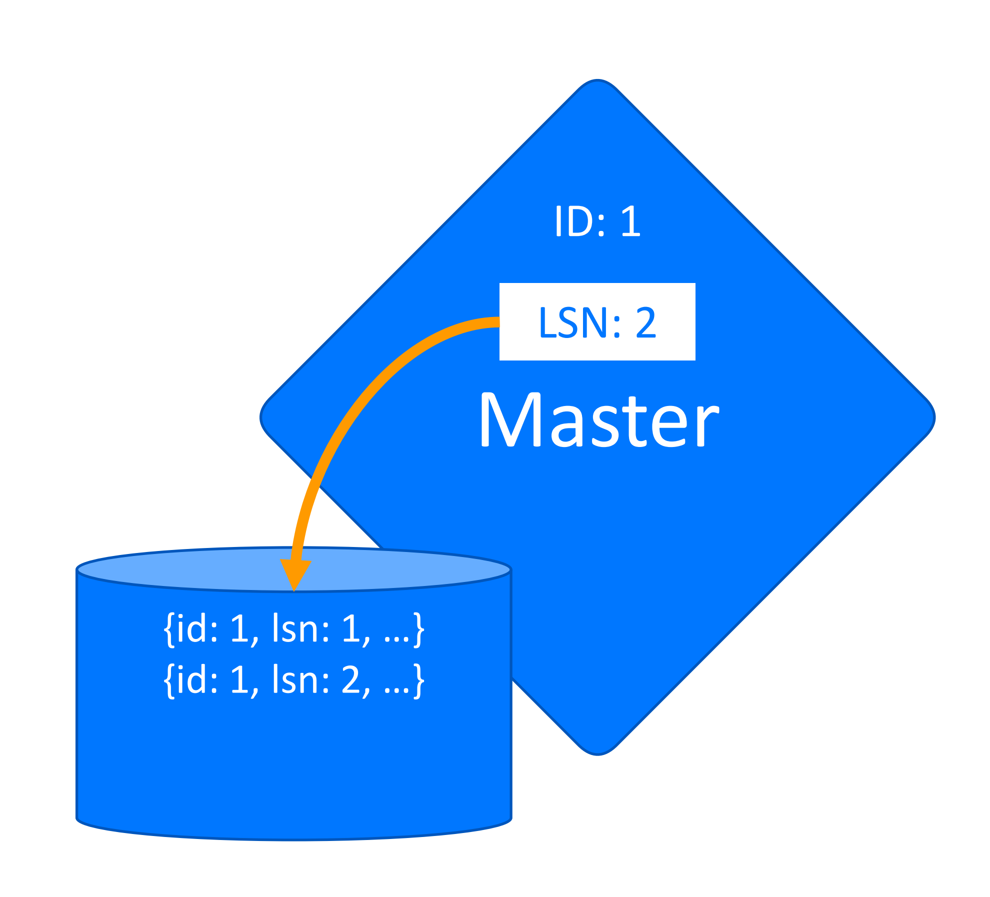
Репликация в Tarantool



Репликация в Tarantool



Репликация в Tarantool



Репликация в Tarantool

ID: 1

LSN: 2

Master

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}

ID: 2

LSN: 0

Replica

Репликация в Tarantool

ID: 1

LSN: 2

Master

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}

ID: 2

LSN: 0

Replica

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}

Репликация в Tarantool

У каждого свой LSN

ID: 1

LSN: 2

Master

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}

ID: 2

LSN: 0

Replica

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}





Репликация в Tarantool

ID: 1

LSN: 3

Master

ID: 2

LSN: 0

Replica

{id: 1, lsn: 1, …}
{id: 1, lsn: 2, …}
{id: 1, lsn: 3, …}

{id: 1, lsn: 1, …}
{id: 1, lsn: 2, …}
{id: 1, lsn: 3, …}







Репликация в Tarantool

ID: 1
LSN: 3
Master

{id: 1, lsn: 1, …}
{id: 1, lsn: 2, …}
{id: 1, lsn: 3, …}

ID: 2
LSN: 0
Master

{id: 1, lsn: 1, …}
{id: 1, lsn: 2, …}
{id: 1, lsn: 3, …}

Репликация в Tarantool

ID: 1
LSN: 3
Master

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}
{id: 1, lsn: 3, ...}

ID: 2
LSN: 1
Master

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}
{id: 1, lsn: 3, ...}
{id: 2, lsn: 1, ...}





Репликация в Tarantool

ID: 1
LSN: 3
Master

ID: 2
LSN: 1
Master

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}
{id: 1, lsn: 3, ...}
{id: 2, lsn: 1, ...}

{id: 1, lsn: 1, ...}
{id: 1, lsn: 2, ...}
{id: 1, lsn: 3, ...}
{id: 2, lsn: 1, ...}





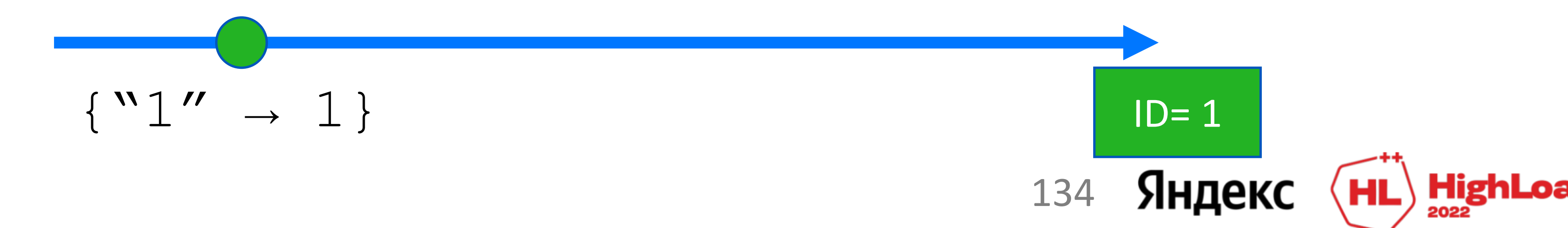
Векторные часы (vclock)

Алгоритм и структура
для упорядочивания событий
в распределённой системе





Векторные часы (vclock)



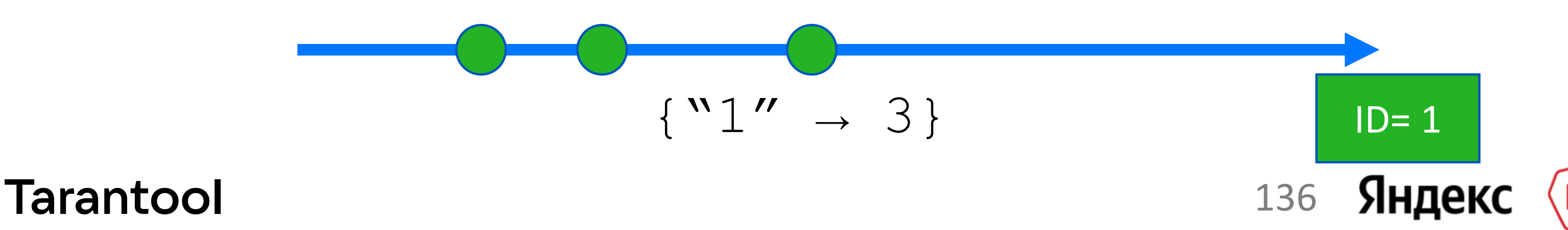


Векторные часы (vclock)

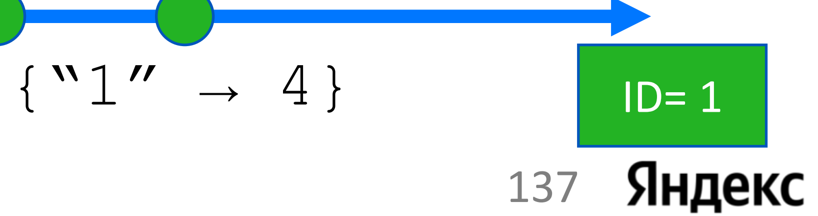




Векторные часы (vclock)



Векторные часы (vclock)

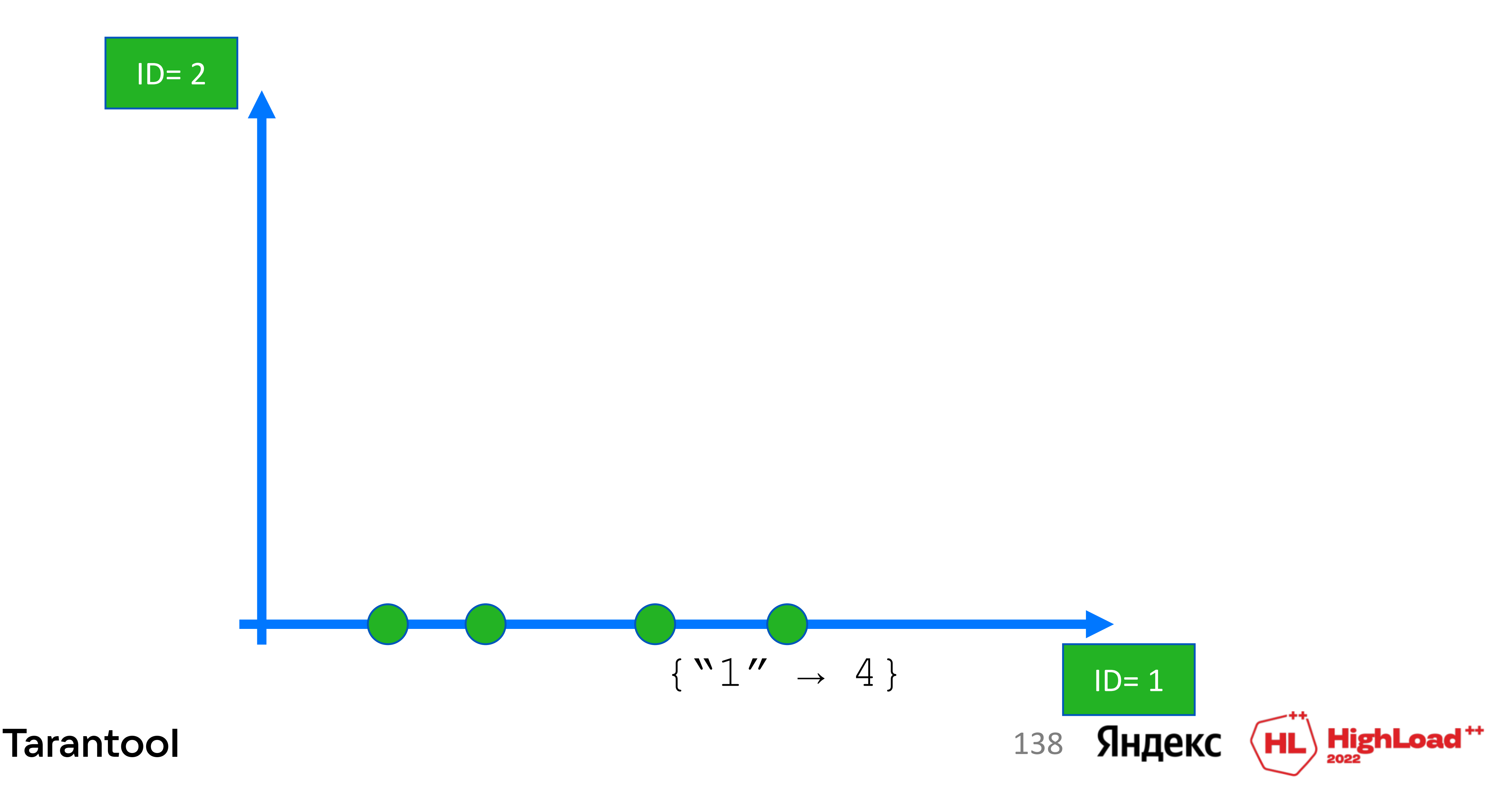




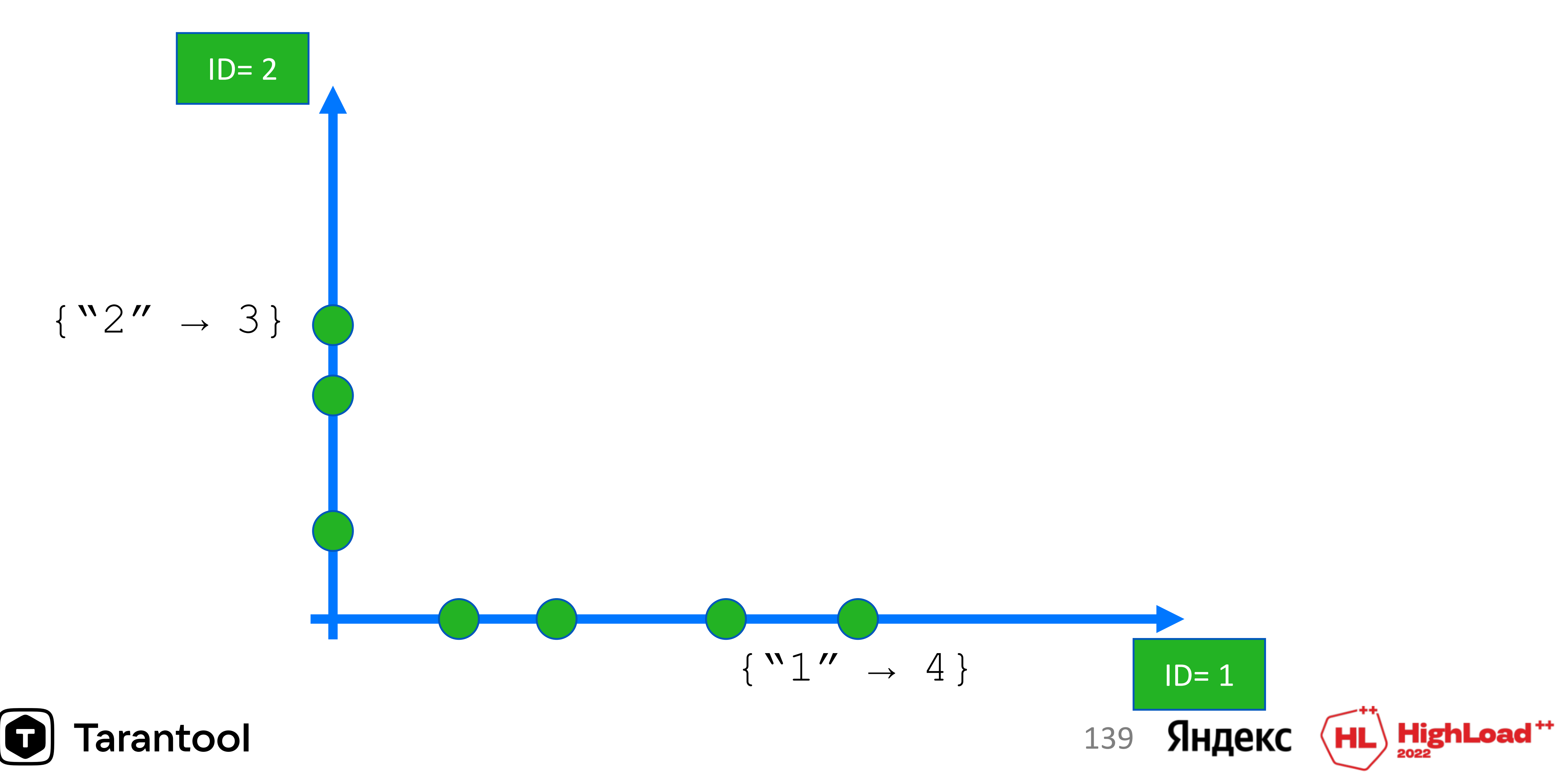




Векторные часы (vclock)



Векторные часы (vclock)



Векторные часы (vclock)

ID= 2

{"2" → 3}

ID= 3

{"3" → 2}

{"1" → 4}

ID= 1







Векторные часы (vclock)

“Up to date” point

{ "1" → 4
"2" → 3
"3" → 2 }

ID= 2

ID= 3

ID= 1

{"2" → 3}

{"3" → 2}

{"1" → 4}





Репликация в Tarantool

- Основана на WAL
- Использует VClock для определения состояния
- Master-Master или Master-Replica
- Произвольная топология
- Синхронная и асинхронная*

* Может быть сконфигурирована per space





05

MVCC

Multi Version Concurrency Control
Борьба за чистоту чтений

Сохранность в памяти

Чтение «не ждёт»

Читаем из памяти

Запись «ждёт»

Пишем в память

Пишем на диск

Сохранность в памяти

Чтение «не ждёт»

Читаем из памяти

Читаем из памяти (“dirty”)

Запись «ждёт»

Пишем в память

Пишем на диск (долго)

Консистентность чтений

RAM

0

Disk





Запись происходит в память

Writer

W

RAM

0

1

Disk





Чтение происходит из памяти

Reader

R

1

Writer

W

RAM

0

1

1

Disk





Запись на диск подтверждается

Reader

R

1

Writer

W

OK

RAM

0

1

1

Disk

1





Запись на диск подтверждается

Reader

Writer

RAM

Disk

W

OK

R

1

R

1

0

1

1

1

1





Некоторые чтения — до записи в WAL

Reader

R 1

R 1

Writer

W

OK

RAM

0 1 1 1

Disk

1





(не)консистентность чтений?

Reader

Writer

RAM

Disk





(не)консистентность чтений?

Reader

Writer

RAM

Disk

R 1 | R 1 | R 2

W | OK | W

0 | 1 | 1 | 1 | 2 | 2

1





Ошибка диска откатывает транзакцию

Reader

Writer

RAM

Disk

R 1 | R 1 | R 2

W | OK | W | X

0 | 1 | 1 | 1 | 2 | 2 | 1

1 | X





Читатель видел «грязные» данные

Reader

Writer

RAM

Disk

R 1 | R 1 | R 2 | R 1

W | OK | W | X

0 1 1 1 2 2 1 1

1 | X





Разделить области чтения и записи







Разделить области чтения и записи

W

RAM 0

RAM 0 1

Disk





Чтение из read-view

R
0
W
RAM
0
0
RAM
0
1
Disk





Чтение из read-view

0
R
W
OK
0
RAM
0
1
RAM
0
1
Disk
1





Чтение только подтверждённого

R 0
R 1
W
OK
0
RAM 0 1 1
RAM 0 1
Disk 1





Чтение только подтверждённого

R 0 R 1

W OK W X

0

RAM 0 0 1 1

RAM 0 1 2

Disk 1 X





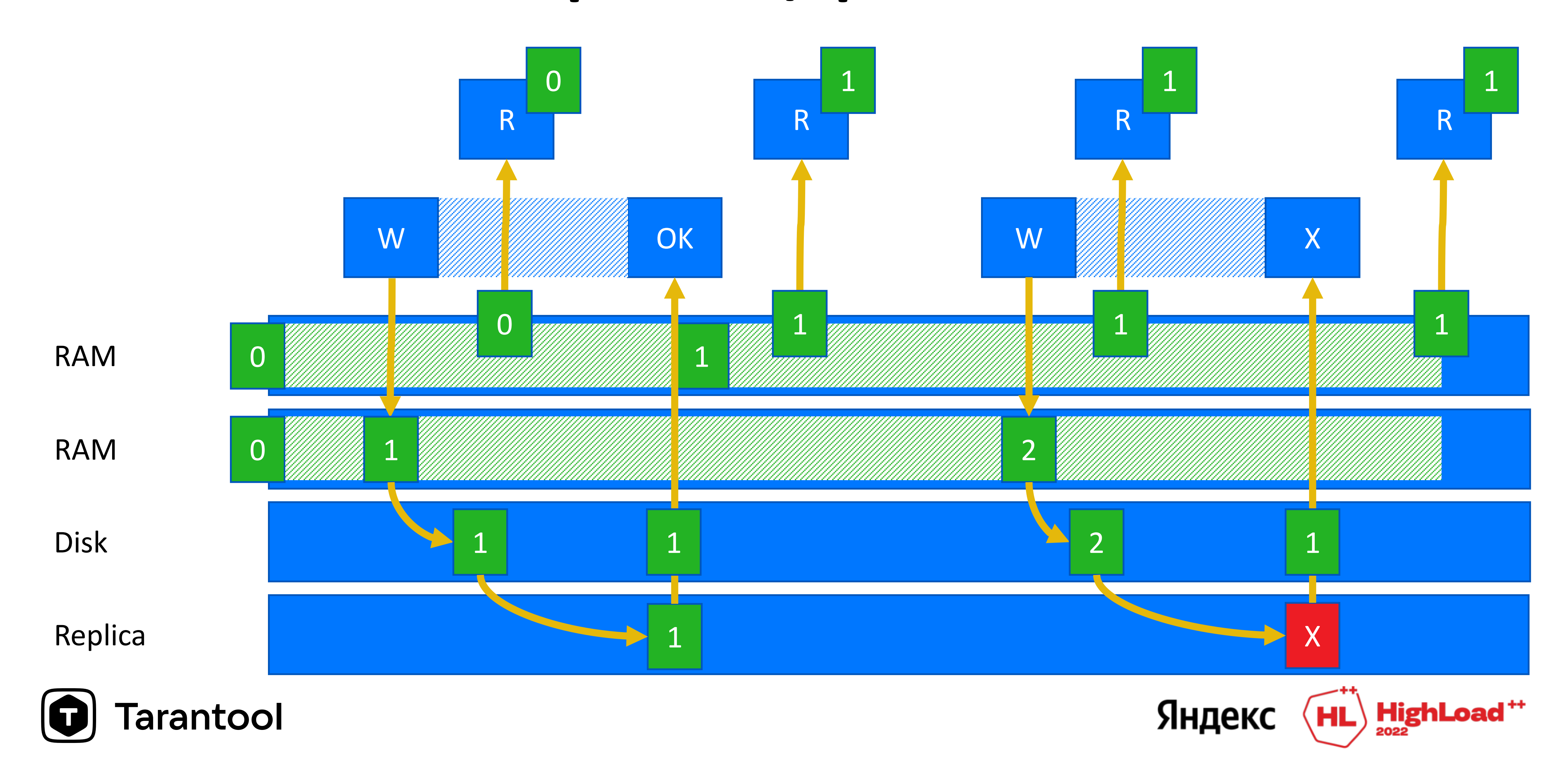
Чтение только подтверждённого

R 0
R 1
R 1
R 1
W
OK
W
X
RAM 0 0 1 1 1 1
RAM 0 1 2
Disk 1 X





Чтение только отреплицированного



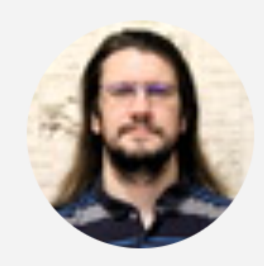
Подробнее об MVCC — завтра

h3: Яндекс трек

25 ноября
(завтра)

14:40 – 15:30

Как работает MVCC
в In-Memory-СУБД



Александр Ляпунов (Tarantool, VK)

Как устроен Tarantool?

- 100% данных находятся в памяти
- Доступ к данным из одного потока
- Доступ к данным только через индекс
- Изменения пишутся во Write Ahead Log (WAL)
- ACID обеспечивается однопоточностью и WAL
- Периодически сохранятся консистентный Snapshot
- WAL реплицируется
- Репликация гибридная (MM, MR, Sync, Async)





Спасибо за внимание!

Вопросы?

bx.vc/un1101

Оставьте отзыв

Скачайте презентацию

bx.vc/xxx

Mons Anderson

inthrax





